

Final Report

Laboratory Information Management Systems,
Center for Technological Diagnostics
Training, Certification, and Calibration

submitted to
European Research Office
USARDSG-UK
Edison House
223 Old Marylebone Road
London, NW1 5TH
England

Laboratory Information Analysis within the Russian Center for
Technological Diagnostics
Broad Agency Research and Development

Reference: N62558-02-C-9041
R&D: 9329-EN-01

Principal investigator:
Dr. J.M.F. Masuch

Institution:
Applied Logic Laboratory (ALL)
Academic Foundation
Amsterdam

Herengracht 514
1017 CC Amsterdam
The Netherlands
Tel: # .31.20.422.9767
Fax: # .31.20.422.9768



15 March 2004

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 15 MAR 2004		2. REPORT TYPE N/A		3. DATES COVERED -	
4. TITLE AND SUBTITLE Laboratory Information Analysis within the Russian Center for Technological Diagnostics				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Applied Logic Laboratory (ALL) Academic Foundation Amsterdam				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited					
13. SUPPLEMENTARY NOTES The original document contains color images.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 192	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Table of Contents

1.0	Introduction	3
2.0	LIMS - Report Generation	5
2.1	Format Functions and Algorithms	7
2.2	Main Execution Procedures	19
2.3	The Graphical User Interface	36
3.0	Algorithm Development and Structured DBMS	44
3.1	PL/SQL Organization and Structure	44
3.2	Cursors and Database Interaction	47
3.3	Adding Parameters to the Cursors	51
3.4	Adding Bind Variables to the Cursor	52
3.5	Applications for Dynamic SQL and DBMS_SQL	54
3.6	Creating Unique Data Records in the LIMS	62
3.7	Applying Array Structures within the LIMS	65
3.8	LIMS Error Functions and Exception Management	66
3.9	LIMS Mail Resources	70
3.10	CTD Web Applications and Network Distribution	73
3.11	CTD Instrumentation and Maintenance Documentation	94
4.0	Certification and Attestation Programs	117
5.0	Level II Field Diagnostics and Attestation Training	132
6.0	Conclusions	147
7.0	References	149
8.0	Appendix A: LIMS Time-Date Formats	155
	Appendix B: Arrays and Temporary Data Storage	162
	Appendix C: Date Algebra	164
	Appendix D: Productivity Functions	168
	Appendix E: SQL Management Functions	174
	Appendix F: SQL Plus Functions	176
	Appendix G: Security Package	178

1.0 Introduction

In this report, we examine the technical parameters for the design and implementation of a Laboratory Information Management System (LIMS) within the Center for Technological Diagnostics (CTD) in St. Petersburg Russia. The LIMS is composed of two main databases. The first database is used to store, archive, retrieve, and manage materials information as it is placed within the StarLIMS software. The StarLIMS interface uses the Sybase Relational Database Manager (RDBMS) as the main archive tool; however, all information is distributed across the Ministry of Defense (MOD) network. The second database contains the archived information (materials data) within the Oracle RDBMS. This data is stored within the CTD main server system. The main server is a Compaq ML-530 array with eight independent disks. Two identical Compaq ML-530 systems are used to archive the technical information and serve all information that is required for the fixed and mobile laboratories. The disks are partitioned for data security, and organized by technical requirements for LIMS access and materials archive.

The LIMS is designed to assist MOD in their certification requirements, and provide chain-of-custody tools that are necessary to maintain proper authorization and control of information as it enters (and exits) the laboratory complex. The LIMS is also required to assist DTRA in examination requirements for monitoring the proper use of all equipment in the functional laboratory. For certification requirements, the LIMS produces specialized reports from each testing sequence. The reports include documented headers that are authorized by the 12th Main Directorate, as well as, specialized data structures and attestations that are required by the Russian State Standards (RSS) and Gosgorteknadzor oversight bureaus.

As provided in this technical manuscript, the LIMS interface uses a common theme (kernel) that ensures the safety and security requirements for the 12th Main Directorate. The security kernel ties all operations to a specific user (test engineer), and his respective abilities to perform a certification or analytical sequence. Hence, only authorized personnel may perform specific tests, and results are organized according to the strict rules and regulations from the Russian Bureau of Mining: Gosgorteknadzor. The security features force all users to perform tests in the proper sequence -- based upon rules and regulations for proper data analysis. Within this sequence, time indexing is used to insure that all examinations are organized in the proper order. This includes: intermediate authorization for all tests with full metadata records, chain-of-custody examination, piecewise tracking for all tests performed in a sequence, and strict-compliance with Gosgorteknadzor and Russian State Standards (RSS) technical reporting. The metadata records include: unique identification codes that label the specimen under investigation, as well as, the name of the analyst and his respective attestation level. The metadata records also include detailed information for the sequence of testing operations used to generate the technical report.

Within each testing sequence, MOD utilizes productivity tools that have been developed within this research and development effort. The tools provide a consistent foundation for migrating information into the LIMS from the Sybase and Oracle RDBMS. In addition, tools have been developed to parse and organize technical data according to the International Organization for Standardization (ISO) requirements. The ISO methods utilize the Structured Query Language (SQL) techniques for importing and exporting information into RDBMS systems within the European Union (EU) and Former Soviet Union (FSU). These methods follow the strict technical standards from the American Materials Testing Institute (AMTI).¹

¹ The software is in compliance with the Microsoft SQL standard, and the Oracle PL/SQL language. Hence, all algorithms may be modified by the 12th Main Directorate to ensure maximum flexibility.

In this report we document the LIMS algorithms and the Graphical User Interface (GUI) standards that have been created to assist MOD in their efforts to analyze material samples acquired from cranes, hoists, lifts, and elevator systems. The analysis is required to ensure the safe and efficient movement of super-containers that contain sensitive weapon systems. The LIMS is also used for the monitoring and diagnostics of high-pressure piping and vessel systems. These systems are used in facilities adjacent to the super-container structures, and may present significant hazards to the site location.

The LIMS GUI begins with a series of *fidelity* checks. The fidelity checking is required to ensure that the acquired sample has been properly processed during field acquisition. The LIMS further checks the user identification codes and the respective abilities or attestation levels that have been assigned to the user for the testing sequence. Once the sample and user are authorized, the LIMS monitors the testing procedures, and ensures proper sequential data processing and scientific investigation. Finally, the LIMS assists the user in the decision process (e.g. assigning the proper analysis methods to the investigative sample), and creates the authorized reports that are required by the MOD 12th Main Directorate for RSS and Gosgorteknadzor.

With the initial section of this report, we provide a series of images and screen-shots that capture the working dynamics of the LIMS process. These images include the security and identification procedures that are used to monitor the users as they apply tests to specific samples. As shown in these case studies, the systems administrator may monitor all users as they process the sample through each test sequence. The LIMS also monitors the a priori tests that have been applied to similar samples and creates the chain-of-custody records that are required for RSS and Gosgorteknadzor attestation. As indicated, the GUI is dynamic. Hence the tables and interface options vary with the testing sequence and the appropriate analysis methods. For example, the user is provided only those interface options that are required to perform the specific non-destructive test. If the user tries to circumvent the standards, the interface will not provide the data I/O that is required to continue the analysis. In addition, the systems administration personnel are shown the sequential operations (in real time) to ensure that the analyst is made aware of the resultant error or is properly authorized to continue the sequence. The GUI is dynamic with respect to the data migration procedures. As the analyst performs the test, the LIMS is automatically populated with the data that has migrated from the instrument to the Sybase/Oracle RDBMS. The migration includes Quality Assurance and Quality Control (QA/QC) analytical methods. These include Pearson, Spearman, and Kendall Split-Half Coefficients to monitor the data reliability. In addition, the user may edit the data as it is migrated into the analysis technique — provided there is sufficient authorization to perform this function. As in all LIMS processing, editorial revisions and data analysis methods are fully documented and become part of the metadata and chain-of-custody record.

Within this documentation, we include a series of records that demonstrate the MOD 12th Main Directorate and RSS/Gosgorteknadzor reporting standards. As shown, the reports have been developed in conjunction with MOD Moscow and approved for submission to RSS/Gosgorteknadzor for sample attestation. Each report includes the sample documentation, the analysis standards, and resultant conclusions for service life determination.

As requested by MOD (LCOL. Trofimov and LCOL. Protopopov), we provide the SQL and PL/SQL support functions that operate within the Sybase and Oracle RDBMS. The functions are required to allow MOD to develop custom processes on an independent basis according to MOD 12th Main Directorate standards for open systems interfaces. These procedures are organized according to basic productivity standards for parsing information, data I/O and data migration, as well as, modern imaging and data analysis requirements. The SQL and PL/SQL support functions will be used to minimize future maintenance requirements and provide an independent method for MOD to fulfill their technical requirements for custom test generation and specialized reporting.

The final section of this report provides technical documentation from the certification and calibration program. The training is required to ensure that MOD field teams operate systems that are properly calibrated and registered for RSS/Gosgorteknadzor attestation. The program also includes specialized training that is conducted for MOD field teams for the certification of test results. In this documentation sequence, the training program for three additional students is provided. Examinations are shown for all students based upon their attestation level (Level I or Level II), as well as, their prior training within this research program.

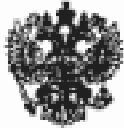
A final addendum to this report will be forwarded to DTRA. This addendum includes digital photos of the LIMS -- operating within the MOD scientific facility in St. Petersburg. The photos have been prepared for the 12th Main Directorate authorization in Moscow. Under the official rules for technical documentation, the photos will be forwarded directly to the US Embassy in Moscow. A second set of photos with additional screen-documentation (for the StarLIMS interface) will be provided to DTRA during the next delegation visit to St. Petersburg.

2.0 Laboratory Information Management Systems – Report Generation

Within this section we provide the source code for the SQL and PL/SQL tools that support the main CTD LIMS. These tools are independent from the StarLIMS shell. Hence, they may be fully modified and edited by MOD to support mission objectives and certification requirements. The code is very detailed and includes all support functions and utilities as shown in Appendix A-G. The algorithms support specialized requirements for secure operations defined by MOD 12th Main Directorate. However, all routines, functions, and procedures are defined using ISO standards for database access and information processing.

Also within this discussion, we provide the source code and graphical user interface examples for the MOD report generation tool. Reports are created using table entries that allow MOD to custom-enter information that originates from within the LIMS or the CTD materials database. The tools are designed for report entry and data query. Hence, the analysts can use the same tool to locate prior reports and enter new information within the LIMS network. The interface for this application is shown in Figure 2.1a. In this example, the basic GUI is displayed with minimum data insertion. The tool is displayed for a specific testing sequence and the corresponding reports that are available within the LIMS network.

From a table driven list of available fields (Fixed or Mobile, Instrument, Item, Measurement), the user selects the entries that will appear in the report by clicking on left and right arrows in the center of the interface. The fields can then be rearranged by choosing the + or – buttons on the left of the application. The header form for the report is inserted by MOD to allow all reports to conform to Russian State Standards or Gosgorteknadzor attestations. The application can be filtered by any value in any of the chosen fields, and sorted in either ascending or descending order. Once the report variables have been chosen, a “Preview Report” option is available to review the query. The preview report columns can be rearranged by dragging the column, and sorted by clicking on the top of a column. Each column may be modified to conform to certification or attestation requirements. Reports may be modified and exported to Microsoft Excel using the “Save Custom Report” option.



12th Main Directorate

Laboratory Information Management System

Testing Sequence	<	>	Report Generation	+	-
<div style="border: 1px solid black; padding: 5px; min-height: 100px;"> Fixed or Mobil Instrument Item Measurement </div>	<	>	<div style="border: 1px solid black; padding: 5px; min-height: 100px;"> *****None Selected***** </div>	+	-
Filter: None ▼	=		Value: None ▼		
Sort: None ▼			Order: Ascending ▼		
<div style="border: 1px solid black; padding: 10px; text-align: center; margin-bottom: 10px;"> Preview Report </div> <div> Available Saved Reports: Select Report ▼ </div>			<div> Save Report As: <div style="border: 1px solid black; height: 20px; width: 100%;"></div> </div> <div style="margin-top: 10px;"> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team <input type="radio"/> RSS Standards <input type="radio"/> Gosgorteknadzor <input type="radio"/> QA/QC Determination </div> <div style="text-align: center; margin-top: 10px;"> Save Custom Report </div>		

Figure 2.1a: LIMS Report Generation Interface. This graphical user interface is used to select custom reports that may be submitted to the attestation agencies: RSS and Gosgorteknadzor. The tool also supports the custom formats required by the fixed and mobile laboratories for the 12th Main Directorate in Moscow.

The complete algorithms for this application are provided in Section 2.1. As shown, this application uses SQL and PL/SQL to build the database requirements and then works within Java, C, or html to support the standard GUI that is displayed to the user. The application conforms to the strict software requirements of the 12th Main Directorate since it is open source code for tempest-level certification and uses International Organization for Standardization techniques for data transfer and software design. The tool is written for efficient modification by MOD. All look and feel portions of this application may be adapted to conform to the future service requirements of the 12th Main Directorate. In addition, the safety and security applications (shown in Appendix G) may be modified to meet the emerging software requirements for secure data processing within the scientific research facility in St. Petersburg.

2.1 Format Functions and Algorithms

Within this section we provide the source code for the SQL and PL/SQL tools that support the main application. All examples have been tested using the standard Oracle 8.i RDBMS for efficient service and support. The interface examples are provided in English for this report. However, the *koi-8* Russian Cyrillic Font is used for all installed software as requested by MOD.

The report generation tool is initialized by variable type and final application. The example declarations include:

```
CREATE TABLE LAB_CUSTOM_REPORTS
(
  USERNAME          VARCHAR2(100 BYTE),
  PRIV_CODE         NUMBER,
  CREATED           DATE,
  REPORT_NAME       VARCHAR2(100 BYTE),
  BASE_REPORT       VARCHAR2(50 BYTE),
  REPORT_COLUMNS    INT ARRAY,
  AVAILABLE_TO      VARCHAR2(100 BYTE),
  REPORT_ID         NUMBER,
  REPORT_SQL        VARCHAR2(4000 BYTE)
)
```

The table for holding all laboratory events is created within the *Lab_Reports* sequence. This table contains the SQL code that generates the data for the final application. The application is dynamically created from this SQL statement.

```
CREATE TABLE LAB_REPORTS
(
  REPORT_NAME       VARCHAR2(100 BYTE),
  REPORT_SQL        CLOB,
  DRILLDOWN         CLOB,
  DRILLDOWN_DOC     CLOB,
  DRILLDOWN_J       CLOB
)
```

Now the reporting table can be populated using sample data. For the example shown in Figure 2.1b, we populate the table with sample information that has been provided by MOD. The actual data is automatically entered into the *Lab_Reports* sequence using the LIMS data input-output (I/O) tools. However, this example shows the minimum level requirements for entering this form of information at the code-level. This also illustrates the simple formatting that is required to insert string and character format information.

In this figure, the case tools for placing information are required to ensure that the data is organized by equipment type and process. For example, the field that inserts information into the *fnstrument* examines the field list *L.FM* and assigns a label "Primary element detection and spectral analysis" when the listed measurement function *L.measurement_fcn* is equal to one. This is the first user selection or first item in the list. The logic continues for each element that is used to populate the table with entries.

The logic shown in Figure 2.1b was provided to MOD to show how SQL tables and models may be organized from within their custom applications. This technique has been used within the Information Analysis System (IAS) for the custom labeling and organization of the Geographic Information System (GIS) reports used for mobile operations. The same methods are used in the CTD mobile operations for the support of SQL data I/O tools that are used for the field entry and initial organization of the materials data.

```

INSERT INTO LAB_REPORTS ( REPORT_NAME, REPORT_SQL, DRILLDOWN, DRILLDOWN_DOC,
DRILLDOWN_J ) VALUES (

'project2', 'SELECT  L.item as "fItem",

case when L.FM='M' THEN 'Mobil' else 'Fixed' end as "fFixed or Mobil",
L.Instrument as "fInstrument",
case when (L.FM='F' and L.measurement_fcn ='1') then
'Primary element detection and spectral analysis'
when (L.FM='F' and L.measurement_fcn ='2') then
'Tensile and compression testing'
when (L.FM='F' and L.measurement_fcn ='3') then
'Cyclic testing for tension, compression and material ductility'
when (L.FM='F' and L.measurement_fcn ='4') then
'Primary hardness detection'
when (L.FM='F' and L.measurement_fcn ='5') then
'Optical measurement of metallographic specimen'
when (L.FM='F' and L.measurement_fcn ='6') then
'Sample preparation'
when (L.FM='M' and L.measurement_fcn ='1') then
'Primary element detection and spectral analysis'
when (L.FM='M' and L.measurement_fcn ='2') then
'NDT-UT (non-destructive testing, ultrasonic) flaw detection'
when (L.FM='M' and L.measurement_fcn ='3') then
'Network analysis using NDT-UT'
when (L.FM='M' and L.measurement_fcn ='4') then
'NDT-UT surface hardness'
when (L.FM='M' and L.measurement_fcn ='5') then
'NDT-UT material thickness'
when (L.FM='M' and L.measurement_fcn ='6') then
'Magnetic particle flaw detection'
when (L.FM='M' and L.measurement_fcn ='7') then
'Surface hardness (pin destructive testing)'
when (L.FM='M' and L.measurement_fcn ='8') then
'Optical measurement metallographic examination'
when (L.FM='M' and L.measurement_fcn ='9') then
'NDT residual stress analysis of sub-surface structure'
when (L.FM='M' and L.measurement_fcn ='10') then
'Pressure detection-vacuum detection'
when (L.FM='M' and L.measurement_fcn ='11') then
'Barometric pressure-hydrographic measurements'
when (L.FM='M' and L.measurement_fcn ='12') then
'Sample preparation and laboratory support'
when (L.FM='M' and L.measurement_fcn ='13') then
'Dynamic torque and acceleration'
when (L.FM='M' and L.measurement_fcn ='14') then
'Capillary and chemical amplification of surface features'
when (L.FM='M' and L.measurement_fcn ='15') then
'Physical resistance or conductance characterization'
when (L.FM='M' and L.measurement_fcn ='16') then
'Magnetic field detection and sub-element characterization'
else ' '
end as "fMeasurement",null as " "
FROM lab_equipment L where 5=5'
, NULL, NULL, NULL);
commit;

```

Figure 2.1b: Within Code Data Population. This example shows one method for populating the report tables based upon MOD case conditions. The logical organization is used to conform to MOD 12th Main Directorate requirements for orienting information by instrument function and analyst certification and attestation. The installed software uses the LIMS main shell to organize this information. Hence, all data is automatically oriented in the proper logical sequence from within the Sybase or the Oracle RDBMS.

Following the data population phase, the system creates a *report_utility* package that works with the RDBMS. This operation creates the cascading style sheets that are used to change the appearance of all the pages in the application from a single source. The procedure is also required to execute the *TwoLists* that are shown in the final GUI. The lists are selected by the user organized field names that will appear in the final MOD report. The form of this declaration is:

```

CREATE OR REPLACE PACKAGE REPORT_UTILITY IS

Procedure TwoLists;
Procedure LoadCSS;
PROCEDURE donothing;
Procedure FilterTable;
Procedure TableSums;
FUNCTION urlencode(p_str in varchar2) return varchar2;

END REPORT_UTILITY;
/

```

The *TwoLists* are then created using the procedures and functions shown in Figure 2.1c,d. This includes the logical statements that are required to automatically sort the data list by the user-defined criteria for instrument time, identification code, or any provided index.

```

CREATE OR REPLACE PACKAGE BODY REPORT_UTILITY as

Procedure TwoLists
is
Begin
  htp.p('
  <script>

sortitems = 1; // Automatically sort items within lists? (1 or 0)

function move(fbox,tbox)
{
  var ct = 0;
  for(var i=0; i<fbox.options.length; i++)
  {
    if(fbox.options[i].selected && fbox.options[i].value != "" &&
    fbox.options[i].value != "-1")
    {
      var no = new Option();
      no.value = fbox.options[i].value;
      no.text = fbox.options[i].text;
      tbox.options[tbox.options.length] = no;
      fbox.options[i].value = "";
      fbox.options[i].text = "";
      ct++;
    }
  }
  if (fbox.options.length == ct)
  {
    var noneSelected='';
    fbox.options[0].value = "-1";
    fbox.options[0].text = "*****None Selected*****";
  }

  BumpUp(fbox);
  if ((sortitems) && (ct > 0))
    SortD(tbox);
}

```

Figure 2.1c: Procedure for Defining Two Lists. This example shows the initial code that is required to setup and define the two lists (testing sequence and report generation) shown in the GUI. The code example continues within Figure 2.1d for the box definitions that are required to sort the context data.

Within Figure 2.1c, the logic is provided for organizing the testing sequence and report generation. This includes the basic definitions that are required to begin the actual sorting operation. The sorting process is continued within Figure 2.1d. In this example, the actual information organized by box operation or *box.options* is sorted using the *SortD* function. This c-code example uses the traditional functions and array scripts to perform this organization. The look and feel of the final output is

```

function BumpUp(box)
{
  for(var i=0; i<box.options.length; i++)
  {
    if(box.options[i].value == "")
    {
      for(var j=i; j<box.options.length-1; j++)
      {
        box.options[j].value = box.options[j+1].value;
        box.options[j].text = box.options[j+1].text;
      }
      var ln = i;
      break;
    }
  }
  if(ln < box.options.length)
  {
    box.options.length -= 1;
    BumpUp(box);
  }
}

function SortD(box)
{
  var temp_opts = new Array();
  var temp = new Object(); var rm = "no";
  for(var i=0; i<box.options.length; i++)
  {
    if ((box.options.length > 0) && (box.options[i].value != -1))
    {
      var temp_opts_entry = new Array();
      temp_opts_entry[0] = box.options[i].text;
      temp_opts_entry[1] = box.options[i].value;
      temp_opts[j] = temp_opts_entry;
      j += 1;
    }
    else
      rm = "yes";
  }
  for(var x=0; x<temp_opts.length-1; x++)
  {
    for(var y=(x+1); y<temp_opts.length; y++)
    {
      if(temp_opts[x][0] > temp_opts[y][0])
      {
        temp = temp_opts[x];
        temp_opts[x] = temp_opts[y];
        temp_opts[y] = temp;
      }
    }
  }

  if (rm == "yes")
    box.options.length -= 1;

  for(var i=0; i<box.options.length; i++)
  {
    box.options[i].value = temp_opts[i][1];
    box.options[i].text = temp_opts[i][0];
  }
}
</script>');
end TwoLists;

```

Figure 2.1d: Ending the Two List Procedure. This example shows the final code that is required for the sorting operation. This allows the user to organize the information in the GUI by ascending or descending order and user defined index value.

defined within the loading of the cascade sheets procedure *LoadCSS*. This operating includes the formatting and indexing requirements that form the basic output. Within this procedure, the font substitutions are made to include the Russian Cyrillic fonts. As shown, the colors and labels may be modified to meet the form requirements of the 12th Main Directorate.

```

Procedure LoadCSS
is
Begin
htp.p('<style type="text/css">');

htp.p('
body {font-family:"Arial"; font-size:"9 pt"; color:Black; }
H1 {font-family:"Arial"; font-size:"20 pt"; color:Navy; }
H2 {font-size:"8 pt"; }
H3 {font-family:"Arial"; font-size:"10 pt"; color:Navy; }
H4 {font-family:"Arial"; font-size:"10 pt"; color:Red; }

P {}
TR {}
TD {color:black; font-size:"8 pt"; }

TH {font-family:"Arial"; font-size:"8 pt"; color:Navy; }
.fld {font-size:"8 pt"; color:Black; background-color:"#a0a0a0"; border:"White Thin";
padding:"2pt"; }
.val {font-size:"8 pt"; color:Black; background-color:"#000022"; margin-left:"0 pt";
margin-right:"0 pt"; border:"White Thin"; padding:"1 pt"; }

.tbsum {font-size:"8 pt"; color:White; background-color:"#064406"; border:"White Thin";
margin-left:"0 pt"; margin-right:"0 pt"; border:"White Thin"; padding:"1 pt"; }

.summ {font_size:"8 pt" color:NAVY; background-color:"#AACAFF"; text-align:"Center";
border:"White Thin"; padding:"1 pt"; margin-left:"0 pt"; margin-right:"0 pt";}

.sumr {font_size:"8 pt" color:NAVY; background-color:"#AACAFF"; text-align:"Right";
border:"White Thin"; padding:"1 pt"; margin-left:"0 pt"; margin-right:"0 pt";}
.tab {font-size:"8 pt"; text-align:"Center"; color:White; background-color:"#0000cc";
padding:"2 pt"; }
.reporttab {font-size:"7 pt"; text-align:"Center"; color:White; background-
color:"#000022"; padding:"2 pt"; }

a {color:navy; font-size:"8 pt";}
.fld {font-size:"8 pt"; color:NAVY; background-color:"#c8c8c8"; border:"White Thin";
padding:"2pt"; }
.fld2 {font-size:"10 pt"; color:WHITE; background-color:"#a0a0a0"; border:"White Thin";
padding:"2pt"; }

.val {font-size:"8 pt"; color:NAVY; background-color:"#c8c8c8"; margin-left:"0 pt";
margin-right:"0 pt"; border:"White Thin"; padding:"1 pt"; }
.hfld {font-size:"8 pt"; color:Navy; background-color:"#DFDFDF"; }

.hfldbold {font-size:"8 pt"; color:Navy; background-color:"#DFDFDF";font-weight:bold; }
.hfld2 {font-size:"10 pt"; color:Navy; background-color:"#DFDFDF"; }

.summ {font_size:"8 pt" color:NAVY; background-color:"#BFE6D0"; text-align:"Center";
border:"White Thin"; padding:"1 pt"; margin-left:"0 pt"; margin-right:"0 pt";}

.sumr {font_size:"8 pt" color:NAVY; background-color:"#BFE6D0"; text-align:"Right";
border:"White Thin"; padding:"1 pt"; margin-left:"0 pt"; margin-right:"0 pt";}
.hval {font-size:"8 pt"; color:Black; background-color:"#E6E6E6";
}

.litegray {font-size:"8.pt"; color:Black; background-color:"#ffffff";text-indent:"4 pt";
padding:"4pt";font-weight:bold; }
.lite {font-size:"8.pt"; color:Black; background-color:"#BFE6D0";text-indent:"4 pt";
padding:"4pt" }
.dark {font-size:"8 pt"; color:Black; background-color:"#A7C6CF";text-indent:"4 pt";
padding:"4pt" }

.litei {font-size:"8.pt"; color:Black; background-color:"#BFE6D0";text-indent:"4 pt";
padding:"4pt"; font-style:"italic"; }
.darki {font-size:"8 pt"; color:Black; background-color:"#A7C6CF";text-indent:"4 pt";
padding:"4pt"; font-style:"italic"; }

.litenp {font-size:"8.pt"; color:Black; background-color:"#BFE6D0";text-indent:"4 pt";}
.darknp {font-size:"8 pt"; color:Black; background-color:"#A7C6CF";text-indent:"4 pt"; }
.tab {font-size:"8 pt"; text-align:"center"; color:White; background-color:"#064406";
padding:"2 pt"; }

```



```

.tab2 {font-size:"8 pt"; text-align:"center"; color:White; background-color:"#066606";
padding:"2 pt"; }

.reporttab {font-size:"7 pt"; text-align:"Center"; color:White; background-
color:"#064406"; padding:"2 pt"; }

.plain {font-size:"8 pt"; color:Black; }
.plainbold {font-size:"8 pt"; font-weight:bold; color:Black; }

.navybold {font-size:"8 pt"; font-weight:bold;color:Navy; }
.navy {font-size:"8 pt";color:Navy; }

.plainboldtext {font-size:"8 pt"; font-weight:bold; color:Black; background-color:White; }
.plainboldtext_big {font-size:"10 pt"; font-weight:bold; color:Navy; background-
color:White; }

.plainwhite {font-size:"8 pt"; color:White; background-color:White; }
.menu {font-size:"8 pt"; text-align:"center"; alink:"#00ff00"; background-color:"#ffffff";
padding:"2 pt";}

.report {font-size:"8 pt"; color:Black; background-color:"#0000ff"; text-align:"Left"; }
.navybg {font-size:"8 pt"; color:White; background-color:"#000066"; text-align:"Center"; }

.sansa {
font-family: Arial, Helvetica, sans-serif
}
.sansb, .sansa small b, .sansa b small, small.sansa b, small b.sansa {
font-family: "Myriad Web", Verdana, Helvetica, Arial, sans-serif
}

.serifa {
font-family: "Minion Web", Georgia, Palatino, "Times New Roman", serif
}
small.sansa,.sansa small {
font-family: Helvetica, sans-serif
}

big.sansb,.sansb big {
font-family: "Myriad Web", Tahoma, Verdana, Helvetica, Arial, sans-serif
}
.min {
font-size: 12px
}

#tree {
font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
font-size: 11px;
}

#tree img {
border: 0px;
width: 19px;
height: 16px;
}
#tree lnk{
color: blue;
text-decoration: underline;
}

');

http.p('</style>');
End LoadCSS;

```

Figure 2.1e: The LoadCSS Procedure. The process completes the look-and-feel declarations. Note the font declarations and form inputs that are used to customize the application for English and Russian.

The table entries are organized in ascending or descending order. However, the user may define custom filters that are used to match specific cases and technical reports. The organization is required to select case studies that comply with specific attestation or certification requirements. For example, MOD may use this technique to extract all reports that comply with Gosgorteknadzor Standard GOS-1456-02. This may be accomplished by parsing through all technical information in the LIMS that is

associated with the respective standard and then displaying these results within a custom report sequence. The filter operation is shown in Figures 2.1f-h. The initial operations are shown in Figure 2.1f for the preliminary sorting characteristics. The logical code is extended in Figure 2.1g,h to include context specific methods for sorting the data by string type and index value.

```

Procedure FilterTable is
Begin
http.p('<script>
function _TF_trimWhitespace(txt) {
    var strTmp = txt;
    //trimming from the front
    for (counter=0; counter<strTmp.length; counter++)
        if (strTmp.charAt(counter) != " ")
            break;
    //trimming from the back
    strTmp = strTmp.substring(counter,strTmp.length);
    counter = strTmp.length - 1;
    for (counter; counter>=0; counter--)
        if (strTmp.charAt(counter) != " ")
            break;
    return strTmp.substring(0, counter+1);
}

function _TF_showAll(tb) {
    for (i=0;i<tb.rows.length;i++)
    {
        tb.rows[i].style.display = "";
    }
}

function _TF_shouldShow(type, con, val) {
    var toshow=true;
    if (type != null) type = type.toLowerCase();
    switch (type)
    {
        case "item":
            var strarray = val.split(",");
            innershow = false;
            for (ss=0;ss<strarray.length;ss++){
                if (con==_TF_trimWhitespace(strarray[ss])){
                    innershow=true;
                    break;
                }
            }
            if (innershow == false) toshow = false;
            break
        case "full":
            if (val!=con) toshow = false;

            break
        case "substring":
            if (val.indexOf(con)<0) toshow = false;
            break

        default: //is "substring1" search
            if (val.indexOf(con)!=0) //pattern must start from 1st char
                toshow = false;
            if (con.charAt(con.length-1) == " ")
            { //last char is a space, so lets do a full search as well
                if (_TF_trimWhitespace(con) != val)
                    toshow = false;
            }
            else
                toshow = true;
        }
    }
    return toshow;
}

```

Figure 2.1f: The FilterTable Procedure (Step 1). The preliminary sorting process with case declarations for the string search operation.

```

function _TF_filterTable(tb, conditions) {
    //given an array of conditions, lets search the table
    for (i=0;i<tb.rows.length;i++)
    {
        var show = true;
        var rw = tb.rows[i];
        for (j=0;j<rw.cells.length;j++)
        {
            var cl = rw.cells[j];
            for (k=0;k<conditions.length;k++)
            {
                var colKey = cl.getAttribute("TF_colKey");
                if (colKey == null) //attribute not found
                    continue; //so lets not search on this cell.
                if (conditions[k].name.toUpperCase() == colKey.toUpperCase())
                {
                    var tbVal = cl.abbr;
                    var conVals = conditions[k].value;
                    if (conditions[k].single) //single value
                    {
                        show = _TF_shouldShow(conditions[k].type,
                            conditions[k].value, cl.abbr);
                    } else { //multiple values
                        for (l=0;l<conditions[k].value.length;l++)
                        {
                            innershow = _TF_shouldShow(conditions[k].type,
                                conditions[k].value[l], cl.abbr);
                            if (innershow == true) break;
                        }
                        if (innershow == false)
                            show = false;
                    }
                }
            }
            if (show == false)
                break;
        }
        if (show == true)
            tb.rows[i].style.display = "";
        else
            tb.rows[i].style.display = "none";
    }
}

function TF_filterTable(tb, frm) {
    var conditions = new Array();
    if (frm.style.display == "none") //filtering is off
        return _TF_showAll(tb);

    var inputs = frm.tags("INPUT");
    for (i=0;i<inputs.length;i++)
    { //looping thru all INPUT elements
        if (inputs[i].getAttribute("TF_colKey") == null) //attribute not found
            continue; //we assume that this input field is not for us
        switch (inputs[i].type)
        {
            case "text":
            case "hidden":
                if(inputs[i].value != "")
                {
                    index = conditions.length;
                    conditions[index].name =
                        inputs[i].getAttribute("TF_colKey");
                    conditions[index].type =
                        inputs[i].getAttribute("TF_searchType");
                    conditions[index].value = inputs[i].value;
                    conditions[index].single = true;
                }
                break
        }
    }
}

```

Figure 2.1g: The FilterTable Procedure (Step 2). The secondary sorting process with array declarations for the string search operation.

```

var inputs = frm.tags("SELECT");
//able to do multiple selection box
for (i=0;i<inputs.length;i++)
{ //looping thru all SELECT elements
  if (inputs[i].getAttribute("TF_colKey") == null) //attribute not found
    continue; //we assume that this input field is not for us
  var opts = inputs[i].options;
  var optsSelected = new Array();
  for (intLoop=0; intLoop<opts.length; intLoop++)
  { //looping thru all OPTIONS elements
    if (opts[intLoop].selected
      && (opts[intLoop].getAttribute("TF_not_used") == null))
    {
      index = optsSelected.length;
      optsSelected[index] = opts[intLoop].value;
    }
  }
  if (optsSelected.length > 0) //has selected items
  {
    index = conditions.length;
    conditions[index] = new Object;
    conditions[index].name = inputs[i].getAttribute("TF_colKey");
    conditions[index].type = inputs[i].getAttribute("TF_searchType");
    conditions[index].value = optsSelected;
    conditions[index].single = false;
  }
}
//ok, now that we have all the conditions, lets do the filtering proper
_TF_filterTable(tb, conditions);
}

function TF_enableFilter(tb, frm, val) {
  if (val.checked) //filtering is on
  {
    frm.style.display = "";
  } else { //filtering is off
    frm.style.display = "none";
  }
  //refresh the table
  TF_filterTable(tb, frm);
}

function _TF_get_value(input) {
  switch (input.type)
  {
    case "text":
      return input.value;
    break
    case "select-one":
      if (input.selectedIndex > -1) //has value
        return input.options(input.selectedIndex).value;
      else
        return "";
    break;
  }
}

//util function that concat two input fields and set the result in the third
function TF_concat_and_set(salText, salSelect, salHidden) {
  var valLeft = _TF_get_value(salText);
  var valRight = _TF_get_value(salSelect);
  salHidden.value = valLeft + valRight;
}
</script>

');
end;

```

Figure 2.1h: The FilterTable Procedure (Step 3). The final sorting process with the assignments shown by condition index.

```

Procedure TableSums
Is begin
http.p('<script>
function CommaFormatted(amount)
{
var delimiter = ","; // replace comma if desired
var a = amount; var i = parseInt(a);
if(isNaN(i)) { return ""; }
var minus = "";
if(i < 0) { minus = "-"; }
i = Math.abs(i);
var n = new String(i); var a = [];
while(n.length > 3)
{
var nn = n.substr(n.length-3);
a.unshift(nn); n = n.substr(0,n.length-3);
}
if(n.length > 0) { a.unshift(n); }
n = a.join(delimiter);
amount = n; amount = minus + amount;
return amount;
}
// end of function CommaFormatted()

function addCurrency( strValue ) {
var objRegExp =new RegExp("/-?[0-9]");
strValue = CommaFormatted(strValue);
return "$" + strValue;
}

function TableSums(tb){
var nCols=tb.rows[0].cells.length;
var columnTotals=new Array(nCols);
for (z=0;z<nCols;z++) {columnTotals[z]=0;}
for (i=0;i<tb.rows.length-2;i++)
{
var rw = tb.rows[i];
if (tb.rows[i].style.display == "")
{
for (j=0;j<rw.cells.length;j++)
{
var cl= rw.cells[j]; //alert(cl.abbr);
var sumKey = cl.getAttribute("tbSum");
if (sumKey==null) continue;
var tbVal = cl.abbr;
if (i==0){
columnTotal[j]=new Number(0);
}
else{
if (sumKey=="C")
{
columnTotals[j]=0;
}
else
{
columnTotals[j]=parseInt(columnTotals[j])+parseInt(tbVal);
//alert(columnTotals[j]);
}
}
}
}
}
var rw = tb.rows[tb.rows.length-2];
for (j=0;j<rw.cells.length;j++)
{ //alert(rw.cells[j].innerHTML);
if (columnTotals[j]!=0)
{ var currValue = addCurrency(columnTotals[j]);
rw.cells[j].innerHTML = currValue;}
}
}
</script>');
end tableSums;

```

Figure 2.1i: The TableSums Procedure. The algorithms are used to support the arithmetic for certain table elements. The arithmetic sum is created and applied to specific linked operations.

Following the filter operations, the procedure *TableSums* is used to total numeric columns in the final MOD report. If a report field is numeric, and labeled as a column to be summed, the report generator automatically places a formatted column total at the bottom of the report. The algorithm is provided in Figure 2.1i. The function *urlencode* takes the string parameter *p_str* and returns an argument where all non-alphanumeric characters, such as spaces, tabs, and special characters, have been replaced with their hexadecimal equivalents in the form %xx. This is necessary when special characters are required in a URL. This is also required for the parsing of labels in the Russian Cyrillic Font. The *urlencod* algorithm is shown in Figure 2.1j.

```
function urlencode(p_str in varchar2) return varchar2 is
    l_tmp varchar2(6000); l_hex varchar2(16) default '0123456789ABCDEF'; l_num
number; l_bad
    varchar2(100) default ' >{}~\~;?@&<#{|^[\`/:=$+'''''; l_char char(1);
begin
    l_bad:=l_bad||chr(16);
    if p_str is null then return null; end if;
    for i in 1 .. length(p_str) loop l_char := substr(p_str, i, 1); if
instr(l_bad, l_char) > 0 then
        l_num := ascii(l_char); l_tmp := l_tmp || '%' || substr(l_hex, mod(trunc
(l_num / 16), 16) + 1, 1)
        || substr(l_hex, mod(l_num, 16) + 1, 1); else l_tmp := l_tmp || l_char; end
if; end loop;
    return l_tmp;
end urlencode;

END;
/
```

Figure 2.1j: The Urlencode Function. The algorithms are used to support hexadecimal equivalents and specialized character functions in English and Russian Cyrillic Languages.

This *xls* procedure enables the application to load the custom report directly into Microsoft Excel. The application accepts an SQL statement *querybuild3* as a parameter. This statement is parsed, executed and read into an Excel spreadsheet by modifying the *mime_header* property. This procedure is also used in the LIMS data I/O functions. In particular this algorithm is applied to all tab delimited data that is stored in an ASCII format prior to data analysis. A modified form of this procedure is used to support the Leica data transfer for the Inverted Microscope System DM-IR (MEF4M) and the Vickers (Sony) MHT-10 Image Processing System. This allows the user to place data in specific Excel fields and place images (from the microscope system) directly below the data spreadsheet. The initial declarations for the *xls* procedure are shown in Figure 2.1k. The main algorithm is provided in Figure 2.1l.

```
procedure xls( querybuild3 in clob Default null,
               keepfirst in varchar2 default null)
is
    p_separator          varchar2(10) default ',';
    l_theCursor          integer default dbms_sql.open_cursor;
    l_columnValue        varchar2(6000);
    l_status             integer;
    l_colCnt             number default 0;
    l_separator          varchar2(10) default ' ';
    l_cnt               number default 0;
    querybuild          clob;
    g_desc_t            dbms_sql.desc_tab;

begin
```

Figure 2.1k: The *xls* Procedure for Moving Data into Excel. The main declarations are shown in this example. The execution statements are provided in Figure 2.1l.

```

querybuild:=utl_url.unescape(querybuild3);
dbms_sql.parse( l_theCursor, '||querybuild||' , dbms_sql.native );

owa_util.mime_header( 'application/vnd.ms-excel', false );

http.p('Content-Disposition: attachment; filename="ReportOutput.xls"');

owa_util.http_header_close;
for i in 1 .. 255 loop
    begin
        dbms_sql.define_column( l_theCursor, i,
                                l_columnValue, 6000 );
        l_colCnt := i;
    exception
        when others then
            if ( sqlcode = -1007 ) then exit;
            else
                raise;
            end if;
    end;
end loop;

http.p('<table border=1>');

dbms_sql.define_column( l_theCursor, 1, l_columnValue,
                        6000 );

dbms_sql.describe_columns( l_theCursor, l_colcnt, g_desc_t );
http.p('<thead>');
http.p('<tr>');
for i in 1 .. g_desc_t.count loop

    if upper(KeepFirst)='Y' then
        http.pn('<td><b>' || g_desc_t(i).col_name || '</b></td>');
    else
        http.pn('<td><b>' || substr(g_desc_t(i).col_name,2,
                                length(g_desc_t(i).col_name)-1) || '</b></td>');
    end if;

    if i=g_desc_t.count then
        http.p('</tr></thead>');
    end if;
end loop;
http.p('<tbody>');
l_status := dbms_sql.execute(l_theCursor);
loop
    exit when ( dbms_sql.fetch_rows(l_theCursor) <= 0 );
    l_separator := ' ';
    http.p('<tr>');
    for i in 1 .. l_colCnt loop
        dbms_sql.column_value( l_theCursor, i,
                                l_columnValue );
        http.pn('<td>' || l_columnValue || '</td>');
        l_separator := p_separator;
    end loop;
    http.p('</tr>');
    l_cnt := l_cnt+1;
end loop;
dbms_sql.close_cursor(l_theCursor);
http.p('</tbody></table>');
http.p('</ss:Worksheet>');

end xls;

END;
/

```

Figure 2.11: The *xls* Procedure for Moving Data into Excel. The main algorithm and database execution statements.

2.2 Main Execution Procedures

The report builder begins with the main event loop and primary execution statements. The main procedures are shown as:

```
CREATE OR REPLACE PACKAGE Reports AS

Procedure ReportBuilderBody(report_type in varchar2 default 'project2');
  Procedure ReportBuilder_Save(Name_Array in owa_util.ident_arr , Value_array in
owa_util.vc_arr );
  Procedure ReportBuilder_Filter(filterName in number default null, report_type in
varchar2 default 'project2');
  Procedure Preview_Report(Name_Array in owa_util.ident_arr , Value_Array in
owa_util.ident_arr );
  Procedure Show_Custom_Report(report_id in number );

END Reports;
/
```

The initial algorithm *ReportBuilderBody* is the starting point for the event sequence. It accepts a parameter *report_type* with a default *project2*. This parameter specifies the *report_sql code* that the application is using. Modifying this parameter would enable the application to construct reports based on different requirements or multiple database tables. The parameter the *report_sql code* is passed into this procedure by the CTD LIMS and the CTD materials database. As a result, custom reports are generated by the LIMS using this algorithm to match the technical requirements for the certification.

The procedure simply sets up the body of the user interface to enable the user to choose options regarding the layout and storage of the report. The declarations are shown as:

```
Procedure ReportBuilderBody (report_type in varchar2 default 'project2')
is
  stmt                                clob;
  stmt3                              varchar2(4000);
  l_theCursor                        integer default dbms_sql.open_cursor;
  l_columnValue                     varchar2(4000) default null;
  l_colCnt                           number default 0;
  l_descTbl                          dbms_sql.desc_tab;
  type gCursorType                   is ref cursor;
  gCursor                            gCursorType;
  text_role                          varchar2(100);
  the_uname                          varchar2(100);
  user_role                          number;
  stmt2                              varchar2(1000);
  rept_name                          varchar2(100);
  rept_id                            number;
  type                               iCursorType is Ref Cursor;
  iCursor                            iCursorType;

begin
```

Following the initialization of the variables *stmt* to *iCursor*, the *stmt3* statement is issued to create the drop down box of available reports. The query populates the drop down and gives the user the ability to retrieve any report that has previously been saved. The assignment is shown in Figure 2.2a with various formatting requirements. The *dbms* fields are used to select the SQL code stored in *lab_reports* and *report_type*. From this information the list of available table columns is populated.


```

stmt3:='SELECT REPORT_NAME,REPORT_ID FROM LAB_CUSTOM_REPORTS';
http.p('

<table width=440 border=0>
<font size="4" face="verdana,arial,times" color="#000066"><strong>

<td></td>

<td align=left><font size="4" face="verdana,arial,times" color="#000066"><strong>12th Main
Directorate</strong></font></td>

</tr><tr>
<td align=center colspan=2><font size="4" face="verdana,arial,times"
color="#000066"><strong>Laboratory Information Management System</strong></font></td>
</tr><tr>
</tr></table>
');

select report_sql into stmt from lab_reports where report_name=report_type;

l_theCursor:= dbms_sql.open_cursor;

dbms_sql.parse(l_theCursor,stmt,dbms_sql.native);
dbms_sql.describe_columns( l_theCursor, l_colCnt, l_descTbl );

for i in 1 .. l_colCnt loop
begin
dbms_sql.define_column( l_theCursor, i,
                        l_columnValue, 2000 );
exception
when others then
if ( sqlcode = -1007 ) then exit;
else
raise;
end if;
end;
end loop;

dbms_sql.define_column( l_theCursor, 1,
                        l_columnValue, 4000 );

REPORT_utility.TwoLists;
REPORT_utility.loadcss;

```

Figure 2.2a: Creating the Pop-Down Menus for the GUI and Initializing the Fields. The algorithm includes the special exceptions that are required to trap error codes and unusual events.

When the page is loaded, the procedures contained in Figure 2.2b-d are called. These procedures construct and operate the lists at the top of the GUI main dialog and support basic window management functions. In addition, they provide the functionality to move items between the lists and modify the order in the selected column. The procedure shown in Figure 2.2b manages the main movement of lists between pop-up (or pop-down) menus, whereas, the algorithms shown in Figure 2.2c-d are required to initialize and support the creation of a new window (as required) to display and support the information that is used during the analysis process. In Figure 2.2b, the *Show_Custom_Report* is called with a *report_id* selected by the analyst or the certification agency. The procedures shown in Figure 2.2c-d are shown in two sections to simplify the case example.

```

http.p('<body onLoad="SortD(document.aform.list1);">');
http.p('<script>
function fnMoveUpDown(oSelect, blnUp) {
if (blnUp) {
if (oSelect.selectedIndex > 0)
oSelect.children(oSelect.selectedIndex).swapNode
(oSelect.children(oSelect.selectedIndex - 1));
}}
else {
if (oSelect.selectedIndex < (oSelect.options.length - 1)) {
oSelect.children(oSelect.selectedIndex).swapNode
(oSelect.children(oSelect.selectedIndex + 1));
}}}
function doPreview(oSelect) {
//alert(oSelect.options.length);

for (i=0;i<oSelect.options.length;i++)
{
oSelect.options[i].selected=true;
//alert(oSelect.options[i].selected);
}

for (i=0;i<oSelect.options.length;i++)
{
if (oSelect.options[i].selected==true)
{
//alert(oSelect.options[i].value);
}
}

document.aform.reportname.disabled=false;
//document.aform.savereport.disabled=false;
document.aform.rt.disabled=false;
for (var b = 0; b < document.aform.rt.length; b++)
document.aform.rt[b].disabled = false;
document.all.thetext.style.color="white";
document.aform.action="!reports.preview_report";
document.aform.report_sql.value="";
document.aform.submit();

}
function setPreview(oSelect) {

if ((oSelect.options.length=="1")&&(oSelect.options[0].value=="-1"))
{
document.aform.previewreport.disabled=true;
document.aform.reportname.disabled=true;
document.aform.savereport.disabled=true;
document.aform.rt.disabled=true;
for (var b = 0; b < document.aform.rt.length; b++)
document.aform.rt[b].disabled = true;
document.all.thetext.style.color="gray";
}
else
{
document.aform.previewreport.disabled=false;
document.aform.reportname.disabled=true;
document.aform.savereport.disabled=true;
document.aform.rt.disabled=true;
document.all.thetext.style.color="gray";
for (var b = 0; b < document.aform.rt.length; b++)
document.aform.rt[b].disabled = true;
}
}
function checkSave() {
if ((document.aform.reportname.value.length==0)|| (document.aform.rt.checked==false)){
document.aform.savereport.disabled=true;
}
else {
document.aform.savereport.disabled=false;
}
}
}

```

Figure 2.2b: Graphical Procedures for Moving Data. These algorithms support the drag-and-drop movements for data fields that operate within the GUI. The *setPreview* function is used to display the preview result to the operator during the edit process.

```

function getReport(reportid){
myWindow=window.open("reports.Show_Custom_Report?report_id="+reportid,"myWindow",
"toolbar=no,location=no,scrollbars=no,status=yes,resizable=yes,location=no,toolbar=no,width
h=500,height=550,top=10,left=10,screeny=25,screenx=50\"");
}

</script>

<style type="text/css">
.graytext {color:gray;}
.normaltext {color:white;}
</style>
');

http.p('<span style="font-family: Tahoma; font-size: 7pt;"><form
name="aform" method="post" action="reports.reportbuilder_save" target="preview">');
http.p('<table class=tab border=1>');

http.p('<tr><td class="tab">Testing Sequence</td><td><br></td>');
http.p('<td class=tab>Report Generation</td><td><br></td></tr>');
http.p('<td class="tab"><SELECT style="font-family: Tahoma; font-size:
7pt;width:185px;" size="10" name="list1" multiple> ');

for i in 1 .. l_colCnt
loop
http.p('<option
value="'||i||'">'||substr(l_descTbl(i).col_name,2,
length(l_descTbl(i).col_name)-1)||'</option>');
end loop;
http.p('</select></td>');

http.p('<td><p>');
http.p('<input type="button" value=">" size="10"
onClick="move(this.form.list1,this.form.list2);setPreview(this.form.list2);"
name="B2">');
http.p('</p><p>');

http.p('<INPUT TYPE="button" VALUE="<" SIZE="10"
onClick="move(this.form.list2,this.form.list1);setPreview(this.form.list2);"
NAME="B1">');
http.p('</p></td>');

http.p('<td class="tab" style="text-align:left;">');
http.p('<SELECT style="font-family: Tahoma; font-size: 7pt;width:185px;"
size="10" name="list2" multiple > ');
http.p('<option value="-1">*****None Selected*****</option>');
http.p('</select></td><td style="text-align:left; font-family: Tahoma;
font-size: 7pt;" >');

http.p('<p><input type="button" value="+" size="7"
onClick="fnMoveUpDown(this.form.list2,true);setPreview(this.form.list2);"
name="B2"></p>');
http.p('<p><input type="button" value="-" size="7"
onClick="fnMoveUpDown(this.form.list2,false);setPreview(this.form.list2);"
name="B2"></p>');

http.p('<input type=hidden name="report_type" value="'||report_type||'">');
http.p('</td></tr>');
http.p('<tr><td class="tab">Filter: <SELECT style="font-family: Tahoma; font-size:
7pt;" size="1" name="filter" width="80%"
onChange="document.all('filterValue').src=
'reports.reportbuilder_filter?filtername='
'+this.options[this.selectedIndex].value"> ');

```

Figure 2.2c: Graphical Procedures for Creating the Preview Window (Part 1). These algorithms support the initial declarations and the look-and-feel operations for the preview window.

```

http.p('<option value="-1">None</option>');
for i in 1 .. l_colCnt
loop
  if substr(l_descTbl(i).col_name,1,1)='f'then
    http.p('<option
value="'||i||'">'||substr(l_descTbl(i).col_name,2,length
(l_descTbl(i).col_name)-1)||'</option>');
    end if;
  end loop;
  http.p('</select></td>');
  http.p('<td class="tab"></td>');
  http.p('<td class="tab" style="text-align:left;" nowrap><iframe marginheight=0
marginwidth=0 width=180 height=25 name="filterValue" frameborder=0
scrolling=no
src="reports.reportbuilder_filter"></iframe></td><br></td></tr>');
  http.p('<tr><td class="tab">Sort: <SELECT style="font-family: Tahoma; font-size:
7pt;" size="1" name="sort" width="80%" > ');
  http.p('<option value="-1">None</option>');

for i in 1 .. l_colCnt
loop
  http.p('<option
value="'||i||'">'||substr(l_descTbl(i).col_name,2,length
(l_descTbl(i).col_name)-1)||'</option>');
  end loop;

  http.p('</select></td>');
  http.p('<td class="tab"><br></td>');
  http.p('<td class="tab" style="text-align:left;">Order: <SELECT style="font-
family: Tahoma; font-size: 7pt;" size="1" name="sortorder" width="80%" > ');
  http.p('<option value="asc">Ascending</option>');
  http.p('<option value="desc">Descending</option>');
  http.p('</td><td><br></td></tr>');

  http.p('</td></tr><tr><td class="tab"><input disabled name="previewreport"
type=button value="Preview Report" onClick="doPreview(this.form.list2);">');
  open iCursor for stmt3;
  http.p('<br><br>Available Saved Reports:<br><SELECT
style="font-family: Tahoma; font-
size: 7pt;" size="1" name="cus_rept" width="80%"
onChange="getReport(this.options[this.selectedIndex].value);" >');
  http.p('<option value="">Select Report</option>');
  loop
    fetch iCursor into rept_name,rept_id;
    exit when iCursor%notFound;
    http.p('<option value="'||rept_id||'">'||rept_name||'</option>');
  end loop;
  close iCursor;
  http.p('</select></td><td><br></td>');

  http.p('<td class=tab style="text-align:left; font-family:
Tahoma; font-size: 7pt;"><span id="thetext" class="graytext">
Save Report As:<br><input width=50 disabled type=text onKeyUp="checkSave();"
name="reportname" value=""><br>');
  http.p('<input name="rt" disabled id="rt" type="radio" value="user" checked
http.p('<td><br></td></tr></table></span>');

  http.p('<iframe class="panel" frameborder=0 style="top:505; height:180;
visibility:visible;" id="preview" name="preview"
src="report_utility.donothing"></iframe>');
  http.p('<input type=hidden name="filterresult" value=""><input type=hidden
name="report_sql" value=""></form>');
  http.p('<style>
  .
  panel{
    position: absolute; align: left; top: 100; left: 0; width: 100%;
    z-index:-1; height: 325; visibility: hidden; font: 12pt Verdana,sans-serif;
    color: navy; border-style: none; margin: 0; padding: 0; overflow: none;
    frameborder: 0; marginheight: 0; marginwidth: 0; border:0; hspace:0;
    vspace:0; scrolling: no;
  }
</style>');
end reportbuilderbody;

```

Figure 2.2d: Graphical Procedures for Creating the Preview Window (Part 2). These algorithms support the final declarations for the preview window. The *Panel* operation is required to determine the size and position of the window based upon the preferences of MOD.

The *ReportBuilder_Filter* procedure constructs the filter values that appear in the drop-down box labeled *Value*. The report is passed the parameter *filterName* by the drop-down box labeled *Filter*. The *filterName* parameter represents a number that corresponds to the filter identification code (generated within the LIMS or the CTD materials database). When the Filter value changes, the procedure

```

Procedure ReportBuilder_Filter(filterName in number default null, report_type in varchar2
default 'project2' )

is
stmt                                varchar2(4000);
stmt3                               varchar2(4000);
fromstmt                            varchar2(4000);
stmt_use                            varchar2(4000);
type selectColType is               varray(4000) of varchar2(4000);
displayCol                          selectColType:=SelectColType();
type                                gcursor_type is ref cursor;
gcursor                             gcursor_type;
the_value                           varchar2(4000);
l_theCursor                         integer default dbms_sql.open_cursor;
l_columnValue                       varchar2(4000) default null;
l_colCnt                            number default 0;
l_descTbl                           dbms_sql.desc_tab;
x                                   number;

begin
    REPORT_utility.loadcss;

    select report_sql into stmt from lab_reports where report_name=report_type;
    stmt3:=stmt;
    x:=1;
    displayCol.extend;
    displayCol(x):=substr(stmt,instr(lower(stmt),'select ')+7,(instr(lower(stmt),' ','')-
        instr(lower(stmt),'select '))-6);
    while (instr(lower(stmt),' ','',1,x)>0)
    loop
        stmt3:=replace(stmt3,substr(stmt3,instr(lower(stmt3),' as '), (instr(stmt3,' ','',1,2)-
            instr(lower(stmt3),' as ')+1)));
        x:=x+1;
        displayCol.extend;
        displayCol(x):=substr(stmt,instr(lower(stmt),' ','',1,x-1)+3,(instr(lower(stmt),' ','',
            1,x)-instr(lower(stmt),' ','',1,x-1))-2);
    end loop;
    if instr(stmt3,'order by ')>0 then
        http.p(x||' - '||instr(lower(stmt3),' order by ')||'<br>');
        http.p(x||' - '|| (length(stmt3)-instr(lower(stmt3),' order by '))||'<br>');
    end if;
    fromstmt:= substr(stmt3,instr(lower(stmt3),'from '),((length(stmt3)+1)-
        instr(lower(stmt3),'from ')));
    if ((filterName is not NULL) and (filterName<>-1)) then
        stmt_use:='select distinct ';
        stmt_use:=stmt_use||displayCol(filterName);
        stmt_use:=stmt_use||' '|| fromstmt;
    end if;
    http.p('<body class="tab" style="font-family: Tahoma; font-size: 7pt; text-align:left;"
topmargin=0 leftmargin=0 marginwidth=0 marginheight=0>');
    http.p('Value: <SELECT style="font-family: Tahoma; font-size: 7pt;" size="1"
name="filter" width="80%"
onChange="parent.document.aform.filterresult.value=this.options[this.selectedIndex].val
ue"> ');
    http.p('<option value="">None</option>');
    if ((filterName is not NULL) and (filterName<>-1)) then
        open gCursor for stmt_use;
        loop
            fetch gCursor into the_value;
            exit when gCursor%NotFound;
            http.p('<option value=""||the_value||">"||substr(the_value,1,20)||"</option>');
        end loop;
    end if;
    http.p('</select>');
end;

```

Figure 2.2e: The *ReportBuilder_Filter* Procedure. These algorithms are used to develop the filters that are displayed as options in the graphical user interface.

returns a list of values that are associated with the selected filter field and then populates the drop down box with values that correspond to the filter request. The algorithm is shown in Figure 2.2e. Within this example, the *report_sql* procedure is used to select the SQL statement associated with the application *report_type*. From this selected information, the values associated with the filter names are generated and displayed on the interface. This gives the user the ability to filter the data to remove unwanted information. The final report is placed within the *ReportBuilder_Save* procedure for archive and storage of the metadata records. This algorithm is shown in Figure 2.2f.

```

Procedure ReportBuilder_Save(Name_Array in owa_util.ident_arr , Value_array in
owa_util.vc_arr )
is
    report_type          varchar2(100);
    theColumns           varchar2(2000);
    theValues            varchar2(2000);
    useCols              int_array:=int_array();
    x                   integer;
    savefor              varchar2(100);
    report_name          varchar2(100);
    the_uname            varchar2(100);
    user_role            number;
    stmt                 varchar2(1000);
    full_username        varchar2(100);
    report_sql           clob;
    type                 iCursorType is Ref Cursor;
    iCursor              iCursorType;
begin
    x:=1;
    for i in name_array.first..name_array.last
    loop
        if name_array(i)='list2' then
            useCols.extend;
            useCols(x):=value_array(i);
            x:=x+1;
        end if;

        if name_array(i)='report_type' then
            report_type:=value_array(i);
        end if;
        if name_array(i)='rt' then
            savefor:=value_array(i);
        end if;
        if name_array(i)='reportname' then
            report_name:=value_array(i);
        end if;
        if name_array(i)='report_sql' then
            report_sql:=utl_url.unescape(utl_url.unescape(value_array(i)));
        end if;
    end loop;

    stmt:='INSERT INTO lab_custom_reports
        (username, priv_code, created, available_to, report_name, base_report,
        report_id, report_sql)
    execute immediate stmt using the_uname, user_role, sysdate, savefor, report_name,
        report_type, report_sql;

    commit;

    stmt:='select full_name from users_tbl where username=:the_uname';
    open iCursor for stmt using the_uname;
    fetch iCursor into full_username;
    close iCursor;
    http.p('<br>Report '||report_name||' created on '||to_char(sysdate,
    'DD-MON-YYYY HH24:MI')||'');
end;
```

Figure 2.2f: The *ReportBuilder_Save* procedure. These algorithms are used to archive the SQL statements that issued the report. The statements include the metadata records for MOD chain-of-custody operations.

The *ReportBuilder_Save* procedure is used to archive the SQL statement that generated the final user report. Since the final statement operates with the RDBMS, this may be used to create the original report provided the database has not changed since the initial execution. For archived information, the *ReportBuilder_Save* procedure will normally display the exact same information with a notation (metadata record) that indicates when the original analysis report was generated. If the report differs from the original published material, the *diff* function may be used to identify the source of the deviation. The *ReportBuilder_Save* procedure accepts flexible input parameters such as *html* forms with a variable number of parameters per query. Procedures called using flexible parameters are prefixed with an exclamation mark (!) in the URL. This indicates that the parameters that follow will be two arrays, a *name_array* and a *value_array*.

The *Preview_Report* procedure is used to construct an SQL statement from the flexible parameter arrays passed to it when the Preview Report button is selected. From this statement the procedure constructs a table for the user to review before saving the report to the file system. The specific operations are shown in Figure 2.2g-i. The declarations for the preview reporting operation is shown as:

```

Procedure Preview_Report(Name_Array in owa_util.ident_arr ,
                        Value_array in owa_util.ident_arr )

is

    stmt                clob default null;
    stmt2               clob default null;
    stmt3               clob default null;
    stmt4               varchar2(14000) default null;
    fromStmt            clob default null;

    l_theCursor          integer default dbms_sql.open_cursor;
    l_columnValue        varchar2(4000) default null;
    l_colCnt              number default 0;
    l_descTbl            dbms_sql.desc_tab;
    ret                  number;
    x                    number;
    test                 varchar2(10) default '10';
    coltype               varchar2(4000);
    tempchar             varchar2(4000);
    typeselectColType    is varray(4000) of varchar2(4000);
    selectCol             selectColtype:=selectColType();
    displayCol            selectColtype:=selectColType();
    selectVal             varchar2(4000);
    type iCursorType     is ref cursor;
    iCursor               iCursorType;
    theColType            varchar2(4000);
    do_drilldown          varchar2(4000);
    startVal              number;
    uniq_id              number;
    keyfield              varchar2(4000);
    i                     number;
    name                  varchar2(4000);
    theColumns            varchar2(4000);
    theValues             varchar2(4000);
    type useColType is   varray(4000) of integer;
    useCols               useColtype:=useColType();
    stmt_use              varchar2(4000);
    filter                number;
    filterValue           varchar2(4000);
    sort                  number;
    sortOrder             varchar2(4000);
    sortPresent           boolean;

    Begin

```

The *Preview_Report* requires a considerable number of arrays to hold the SQL statements and archived information. The actual processing begins with the management of the *value_arrays* that store filters and instrument references:

```

for i in name_array.first..name_array.last loop
    if name_array(i)='list2' then
        useCols.extend;
        useCols(x):=value_array(i);
        x:=x+1;
    end if;

    if name_array(i)='report_type' then
        name:=value_array(i);
    end if;

    if name_array(i)='filterresult' then
        filterValue:=value_array(i);
    end if;

    if name_array(i)='filter' then
        filter:=value_array(i);
    end if;

    if name_array(i)='sortorder' then
        sortorder:=value_array(i);
    end if;

    if name_array(i)='sort' then
        sort:=value_array(i);
    end if;

select report_sql into stmt from lab_reports where report_name=NAME;

stmt3:= stmt;
x:=1;
selectCol.extend;
selectCol(x):= substr(stmt,instr(lower(stmt),'select ')+7,
    (instr(lower(stmt),' as ')-instr(lower(stmt),'select '))-7);

displayCol.extend;
displayCol(x):=substr(stmt,instr(lower(stmt),'select ')+7,
    (instr(lower(stmt),' ','-)-instr(lower(stmt),'select '))-6);
--http.p(displayCol(x)||'<br>');
while (instr(lower(stmt),' ','1,x)>0)
loop
    --http.p(selectCol(x)||'<br>');
    stmt3:=replace(stmt3,substr(stmt3,instr(lower(stmt3),'
        as '), (instr(stmt3,'"','1,2)-instr(lower(stmt3),' as ')+1)));
    x:=x+1;
    selectCol.extend;
    selectCol(x):=substr(stmt,instr(lower(stmt),' ','1,x-1)+3, (instr(lower(stmt),'
        as ','1,x)-instr(lower(stmt),' ','1,x-1))-3);

    displayCol.extend;
    displayCol(x):=substr(stmt,instr(lower(stmt),' ','1,x-1)+3,
        (instr(lower(stmt),' ','1,x)-instr(lower(stmt),' ','1,x-1))-2);

    --http.p(displayCol(x)||'<br>');
    -- http.p(x||' - '||stmt3||'<br>');
    --
    -- http.p(x||' - '||instr(lower(stmt3),' as ')||'<br>');
    -- http.p(x||' - '||((instr(stmt3,'"','1,2)-instr(lower(stmt3),' as '))||'<br>');
    if x=50 then exit;
    end if;
end loop;
if instr(stmt3,'order by ')>0 then
    stmt3:=replace(stmt3,substr(stmt3,instr(lower(stmt3),'order by '), (length(stmt3)-
    instr(lower(stmt3),'order by '))));
    -- http.p(x||' - '||instr(lower(stmt3),' order by ')||'<br>');
    -- http.p(x||' - '||((length(stmt3)-instr(lower(stmt3),' order by '))||'<br>');
    -- http.p(stmt3||'<br>');
    end if;
fromStmt:= substr(stmt3,instr(lower(stmt3),'from '), ((length(stmt3)+1)-
    instr(lower(stmt3),'from ')));

```

Figure 2.2g: The *Preview_Report* Procedure (Step 1). These algorithms are used to organize the digital information, by array format, prior to the actual display sequence.


```

if filter>-1 then
    fromStmt:=fromStmt || ' and ' || selectCol(filter) || ' = '' || filtervalue || ''';
end if;

sortPresent:=false;

if sort>-1 then
    for i in useCols.first..useCols.last
        loop
            if useCols(i)=sort then
                sortPresent:=True;
            end if;
        end loop;
    if Not SortPresent then
        useCols.extend;
        useCols(useCols.last):=sort;
    end if;
end if;

stmt_use:='select distinct ';
for i in useCols.first..useCols.last
    loop
        stmt_use:=stmt_use || displayCol(useCols(i)) || ', ';
    end loop;
stmt_use:=substr(stmt_use,1,length(stmt_use)-2);
stmt_use:=stmt_use || ' from stmt;

if sort>-1 then
    stmt_use:=stmt_use || ' order by ' || selectCol(sort) || ' ' || sortorder;
end if;

Report_utility.loadcss;
Report_utility.FilterTable;
Report_utility.TableSums;

http.p('<form name="filter" onsubmit="TF_filterTable(reportTable,filter);return
false" onReset="_TF_ShowAll(reportTable)">');

l_theCursor:= dbms_sql.open_cursor;

    dbms_sql.parse(l_theCursor,stmt_use,dbms_sql.native);
    dbms_sql.describe_columns( l_theCursor, l_colCnt, l_descTbl );

for i in 1 .. l_colCnt loop
    begin
        dbms_sql.define_column( l_theCursor, i,
                                l_columnValue, 2000 );

        exception
        when others then
            if ( sqlcode = -1007 ) then exit;
            else
                raise;
            end if;
        end;
    end loop;

    dbms_sql.define_column( l_theCursor, 1,
                            l_columnValue, 4000 );

    http.p('<BODY
onLoad="parent.document.aform.report_sql.value='' || utl_url.escape
((report_utility.urlencode(stmt_use)) || ''';TableSums(reportTable);">');

    http.p('<TABLE id="reportTable" style="behavior:url(tableAct.htc);BORDER:
black 1px solid; WIDTH: 70%; font-size : 7pt; background-color:#bbd6bb;"
borderColor=#999999 cellSpacing="0" cellPadding="0" border=1
dragcolor='gray' slcolor=#88ff88 hlcolor=#eeeecc >');

```

Figure 2.2h: The *Preview_Report* Procedure (Step 2). These algorithms are used to perform the main database parsing operations with preliminary cursors.

```

ret:= dbms_sql.Execute(l_theCursor);
do_drilldown:='F';

loop
  if x=0 then
    http.p('<THEAD>');
    http.p('<TR class=reporttab>');
    for i in 1..l_colCnt
      loop
        http.p('<td width=80 align="center" class=tab
          style="font-family: Tahoma; font-size: 7pt;">');
        coltype:=upper(substr(l_descTbl(i).col_name,1,1));
        if coltype='F' then
          http.p('<select style="font-family: Tahoma; font-size:
            7pt;" TF_searchType="full"
            TF_colKey="'||substr(l_descTbl(i).col_name,2,
              length(l_descTbl(i).col_name)-1)||' "
            onChange="TF_filterTable(reportTable,filter);
              TableSums(reportTable);">');

          http.p('<option SELECTED TF_not_used
            value=">"||substr(l_descTbl(i).col_name,2,
              length(l_descTbl(i).col_name)-1)||'</option>');
          http.p('<option value="0">Blanks</option>');
          stmt4:='select distinct '||selectCol(useCols(i))||',
            substr(dump('||selectCol(useCols(i))||'),5,2)'||fromstmt;
          open iCursor for stmt4;

          loop
            fetch iCursor into selectVal, theColType;
            exit when iCursor%NotFound;
            case theColType
              when '12' then http.p('<option
                else http.p('<option value='||selectVal||'>'
                  ||substr(selectVal,1,20)||'</option>');
            end case;
          end loop;

          close iCursor;
          http.p('</select>');
        else
          http.p(substr(l_descTbl(i).col_name,2,
            length(l_descTbl(i).col_name)-1));
        end if;
      http.p('</td>');
    end loop;
  http.p(' </TR></THEAD> '); http.p('<TBODY>');
  end if;
  if (dbms_sql.FETCH_ROWS(l_theCursor))>0 then
    x:=x+1;
    http.p('<tr >');

    for i in 1..l_colCnt
      loop
        dbms_sql.COLUMN_VALUE(l_theCursor,i,l_columnValue);
        coltype:=upper(substr(l_descTbl(i).col_name,1,1));
        --http.p('ColType = '||coltype||'<br>');
        case coltype
          when 'C' then
            if l_columnValue is NULL then
              http.p('<td tbSum="S"
                else
              http.p('<td tbSum="S"
                TF_colKey="'||substr(l_descTbl(i).col_name,2,
                  2,length(l_descTbl(i).col_name)-1)||' "
                abbr='||l_columnValue||'
                align=right>'||to_char(to_number(l_columnValue),
                  'fm1999,999,999,999,999.00')||'</td>');
            end if;
          when 'D' then
            if l_columnValue is NULL then http.p('<td
              TF_colKey="'||substr(l_descTbl(i).col_name,2,
                length(l_descTbl(i).col_name)-1)||' " abbr=0 align=right>');
            http.p(' - </td>');

```

```

else
  http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
  1)||'" abbr='||to_char(to_date(l_columnValue),'J')||'
  align=right>'||to_char(to_date(l_columnValue),'DD-MON-YYYY')||'</td>');
end if;

when 'S' then
  if (l_columnValue is NULL or l_columnValue=' ' or l_columnValue like ' %') then
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr='0'' align=left>');
    http.p(' - </td>');
  else
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr='||l_columnValue||' align=left>'||l_columnValue||'</td>');
  end if;

when 'F' then
  if (l_columnValue is NULL or l_columnValue=' ' or l_columnValue like ' %') then
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr='0'' align=left>'); http.p(' - </td>');
  else
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr='||l_columnValue||' align=left>'||l_columnValue||'</td>');
  end if;

when 'N' then
  if l_columnValue is NULL then
    p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1)||'
    abbr=0 align=right>'); http.p(' - </td>');
  end if;

when 'B' then
  if l_columnValue is NULL then
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr=0 align=right>'); http.p(' - </td>');
  else
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
    1)||'" abbr='||l_columnValue||' align="left" valign="top">');
    src="||package_init.Schema||.utility.deliver_media?graphics_name=BAR_RIGHT.GIF"
    alt="">></font>');
  end if;

http.p(l_columnValue||'</td>');
end if;

else
  http.p('<td '||(l_descTbl(i).col_name)||' align=right>'); http.p(' </td>');
end case;

end loop; http.p('</tr>');

else exit;
end if;
end loop;

http.p('</tbody><tfoot><tr>');

for i in 1 .. l_colCnt loop
  http.p('<td align=right>&nbsp;</td>');
end loop; http.p('</tr><tr>');

http.p('<td COLSPAN='||l_colCnt||'>Report Generated on '||to_char(sysdate,'DD-MON-YYYY
HH24:MI:SS')||'</td></tr>'); http.p('</tfoot></table>');

end preview_report;

```

Figure 2.2i: The *Preview_Report* Procedure (Step 3). These algorithms are used to manage the final graphics and GUI look-and-feel operations that define the appearance of the preview report.

The *Show_Custom_Report* is used to generate the saved report that was previously archived by the LIMS operation. This utility reproduces the local report that was saved by the user during the testing sequence, and may be used to generate archived reports from the CTD materials database.

The procedure uses the *report_id* that defines the report format and the unique identification code for the testing sequence. The declarations for the reporting operation as shown as:

```

Procedure Show_Custom_Report(report_id in number )

is

    stmt          clob;
    stmt2         clob;
    stmt3         clob;
    stmt4         varchar2(14000);
    fromStmt      varchar2(14000);

    l_theCursor   integer default dbms_sql.open_cursor;
    l_columnValue varchar2(4000) default null;
    l_colCnt      number default 0;
    l_descTbl     dbms_sql.desc_tab;

    ret          number;
    x            number;
    test         varchar2(10) default '10';
    coltype      varchar2(4000);
    tempchar     varchar2(4000);

    type selectColType is varray(4000) of varchar2(4000);
    selectCol      selectColType:=selectColType();
    displayCol     selectColType:=selectColType();
    selectVal      varchar2(4000);
    type iCursorType is ref cursor;
    iCursor        iCursorType;

    theColType     varchar2(4000);
    do_drilldown   varchar2(4000);
    startVal       number;
    uniq_id        number;
    keyfield       varchar2(4000);
    i              number;

    thename        varchar2(4000);
    theColumns     varchar2(4000);
    theValues      varchar2(4000);
    useCols        int_array;

    report_name    varchar2(4000);
    stmt_xl        varchar2(4000);
    stmt_use       varchar2(4000);
    begin

```

The *Show_Custom_Report* operations are provided in Figure 2.2j-m. These include the main algorithms for the organization of table data, linked lists, and context specific information. The reports are organized according to the technical requirements of the 12th Main Directorate, including: fields and arrays that are established for each instrument within the fixed and mobile laboratory system. The arrays are used to index information from the vendors as well as data that is required for service, maintenance, and support of the facility. The records include the technical data from RSS and Gosgorteknadzor, as well as, the metadata records required to support the chain-of-custody processing.

```

stmt4:='SELECT r.base_report, r.report_sql, r.report_name
        FROM lab_custom_reports r
        WHERE r.report_id = :report_id';

open iCursor for stmt4 using report_id;
fetch iCursor into thename, stmt, report_name;
close iCursor;

http.p('<script>

        function openExcel(){

                document.filter.onSubmit="";
                document.filter.method="post";
                document.filter.action="report_utility.xls";
                document.filter.target="_blank";
                document.filter.submit();

        }
</script>');

stmt3:= stmt;
x:=1;
selectCol.extend;

selectCol(x):= substr(stmt,instr(lower(stmt),'select
distinct')+16,(instr(lower(stmt),' as ')-
instr(lower(stmt),'
select distinct '))-15);

displayCol.extend;

displayCol(x):=substr(stmt,instr(lower(stmt),'select')+7,
(instr(lower(stmt),'-)-instr(lower(stmt),'select '))-6);

while (instr(lower(stmt),'-',1,x)>0)
loop

        x:=x+1;
        selectCol.extend;
        selectCol(x):=substr(stmt,instr(lower(stmt),'-',1,x-1)+3,
        (instr(lower(stmt),' as ',1,x)
        -instr(lower(stmt),'-',1,x-1))-3);

        displayCol.extend;
        displayCol(x):=substr(stmt,instr(lower(stmt),'-',1,x-1)+3,
        (instr(lower(stmt),'-',1,x)
        -instr(lower(stmt),'-',1,x-1))-2);

        if x=50 then exit;
        end if;

end loop;

if instr(stmt3,'order by')>0 then
        stmt3:=replace(stmt3,substr(stmt3,instr(lower(stmt3),'order by'),
        ((length(stmt3)+1)-instr(lower(stmt3),'order by'))));

end if;
fromStmt:= substr(stmt3,instr(lower(stmt3),'from '),
        ((length(stmt3)+1)-instr(lower(stmt3),'from')));

REPORT_utility.loadcss;
utility.FilterTable;
utility.TableSums;

```

Figure 2.2j: The *Show_Custom_Report* Procedure (Step 1). These algorithms are used to manage the initial declarations for creating the report based using the *report_id* data reference.

```

http.p('<html><head><title>Report Builder</title> ');
http.p('
<table width=480 border=0>
<font size="4" face="verdana,arial,times" color="#000066"><strong>
<td><img vspace=0 hspace=0 width=80 height=80 align=left
</tr><tr>

<td align=center colspan=2><font size="4" face="verdana,arial,times"
color="#000066"><strong>Laboratory Information Management System</strong></font></td>
</tr><tr>
</tr></table>
');

http.p('<form name="filter" onsubmit="TF_filterTable(reportTable,filter);return
false" onReset="_TF_ShowAll(reportTable)">');
http.p('<input type="hidden" name="keepfirst" value="false">');

http.p('<input type="hidden" name="querybuild3" value="'|| utility.urlencode(stmt)||'><input
type=button value="Download Report To MS Excel" onClick="openExcel();">');

    l_theCursor:= dbms_sql.open_cursor;

        dbms_sql.parse(l_theCursor,stmt,dbms_sql.native);
        dbms_sql.describe_columns( l_theCursor, l_colCnt, l_descTbl );

    for i in 1 .. l_colCnt loop
        begin
            dbms_sql.define_column( l_theCursor, i,
                                    l_columnValue, 2000 );
            exception
            when others then
                if ( sqlcode = -1007 ) then exit;
                else
                    raise;
                end if;
            end;
        end loop;

        dbms_sql.define_column( l_theCursor, 1,
                                l_columnValue, 4000 );

http.p('<BODY onLoad="TableSums(reportTable);">');
http.p('<TABLE id="reportTable"
style="behavior:url(utility.deliver_utilityfile?filename=tableAct.htc);BORDER: black 1px
solid; WIDTH: 50%; font-size : 7pt; background-color:#bbd6bb;"
borderColor=#999999 cellSpacing="0" cellPadding="0" border=1 dragcolor='gray'
slcolor=#eeeecc hlcolor=#eeeecc >');

    x:=0;
    ret:= dbms_sql.Execute(l_theCursor);

    do_drilldown:='F';

    loop
        if x=0 then
            http.p('<THEAD>');
            http.p('<TR class=reporttab>');
            if do_drilldown='T' then
                startVal:=2;
                http.p('<td class=tab width=20 align="right">Detail</td>');
            else
                startVal:=1;
            end if;

            for i in 1 .. l_colCnt
                loop

```

Figure 2.2k: The *Show_Custom_Report* Procedure (Step 2). These algorithms are used to manage the database query techniques using the function *dbms_sql.define*.

```

http.p('<td width=80 align="center" class=tab style="font-family: Tahoma; font-size:
7pt;">');
coltype:=upper(substr(l_descTbl(i).col_name,1,1));

if coltype='F' then
http.p('<select style="font-family: Tahoma; font-size: 7pt;" TF_searchType="full"
TF_colKey="'||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1)||' "
onChange="TF_filterTable(reportTable,filter);TableSums(reportTable);">');

http.p('<option SELECTED TF_not_used
value=">' || substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1) || '</option>');
http.p('<option value="0">Blanks</option>');

stmt4:='select distinct ' || selectCol(i) || ', substr(dump(' || selectCol(i) || '),5,2)
' || fromstmt;

open iCursor for stmt4;
loop
fetch iCursor into selectVal, theColType;
exit when iCursor%NotFound;

case theColType
when '12' then http.p('<option value=' || to_char(to_date(selectVal,'DD-MON-
YY'),'J') || '>' || selectVal || '</option>');
else http.p('<option value=' || selectVal || '>' || substr(selectVal,1,20) || '</option>');
end case;

end loop;
close iCursor;
http.p('</select>');

else
http.p(substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1));

end if;
http.p('</td>');
end loop;

http.p(' </TR></THEAD> ');
http.p('<TBODY>');
end if;

if (dbms_sql.FETCH_ROWS(l_theCursor))>0 then x:=x+1;

http.p('<tr >');
for i in 1..l_colCnt
loop

dbms_sql.COLUMN_VALUE(l_theCursor,i,l_columnValue);
coltype:=upper(substr(l_descTbl(i).col_name,1,1));

case coltype
when 'C' then
if l_columnValue is NULL then
http.p('<td tbSum="S"
TF_colKey="'||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1)||' " abbr="0"
align=right>');

http.p(' - </td>');
else
http.p('<td tbSum="S"
TF_colKey="'||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-1)||' "
abbr="'||l_columnValue||' "
end if;

when 'D' then
if l_columnValue is NULL then
http.p('<td TF_colKey="'||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||' " abbr="0" align=right>');

```

Figure 2.21: The *Show_Custom_Report* Procedure (Step 3). These algorithms are used to manage the table descriptions and the graphical layout functions (sequence modeling).

```

    http.p(' - </td>');
  else
    http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="||to_char(to_date(l_columnValue),'J')||'"
align=right>||to_char(to_date(l_columnValue),'DD-MON-YYYY')||'</td>');
  end if;

  when 'S' then
    if (l_columnValue is NULL or l_columnValue=' ') then
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="0" align=left>');
      http.p(' - </td>');
    else
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="||l_columnValue||'" align=left>||l_columnValue||'</td>');
    end if;

  when 'F' then
    if (l_columnValue is NULL or l_columnValue=' ') then
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="0" align=left>'); http.p(' - </td>');
    else
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="||l_columnValue||'" align=left>||l_columnValue||'</td>');
    end if;

  when 'N' then
    if l_columnValue is NULL then
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="0" align=right>'); http.p(' - </td>');
    else
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="||l_columnValue||'" align=right>||l_columnValue||'</td>');
    end if;

  when 'B' then
    if l_columnValue is NULL then
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="0" align=right>'); http.p(' - </td>');
    else
      http.p('<td TF_colKey=""||substr(l_descTbl(i).col_name,2,length(l_descTbl(i).col_name)-
1)||'" abbr="||l_columnValue||'" align="left" valign="top">');
      http.p('<font size="1"><img vspace=0 hspace=0 height=16 alt=""><img vspace=0 hspace=0
height=16
src=""||package_init.Schema||'.utility.deliver_media?graphics_name=REDBAR_RIGHT.GIF"
alt=""></font>');
      http.p(l_columnValue||'</td>');
    end if;
  end case;
end loop; http.p('</tr>');

else
  exit;
end if;
end loop;

http.p('</tbody><tfoot><tr>');
for i in 1 .. l_colCnt
  loop
    http.p('<td class=tab2 align=right>&nbsp;</td>');
  end loop;
http.p('</tr><tr>');

http.p('<td COLSPAN='||l_colCnt||'>Report Generated on '||to_char(sysdate,'DD-MON-YYYY
HH24:MI:SS')||'</td></tr>');
http.p('</tfoot></table>');
end show_custom_report;
END Reports;
/

```

Figure 2.2m: The *Show_Custom_Report* Procedure (Step 4). These algorithms are used to manage the graphical layout for the table and listing services.

2.3 The Graphical User Interface

The report generation tool provides a foundation for MOD to organize their case specific information in a format that is required to support the certification and attestation requirements from RSS and Gosgorteknadzor. The interface also supports the storage archive and retrieval of digital information from the MOD library services and archive facilities. As shown in Figure 2.3a, the user may select cases based upon their location within the MOD facility (fixed or mobile operations) and organize the case specific information by ascending (or descending order).

12th Main Directorate

Laboratory Information Management System

Testing Sequence	Report Generation
<div></div>	<div>Item Instrument Measurement Fixed or Mobil</div>
Filter: None	Value: None
Sort: None	Order: Ascending
<div>Preview Report</div> <div>Available Saved Reports: Select Report</div>	<div>Save Report As: <div></div><div><input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team</div><div><input type="radio"/> RSS Standards</div><div><input type="radio"/> Gosgorteknadzor</div><div><input type="radio"/> QA/QC Determination</div><div>Save Custom Report</div></div>

Figure 2.3a: Accessing the Main Graphical Interface. In this illustration, the MOD analyst is beginning a new query that requires data for instrumentation in either the fixed or mobile laboratory complex. The interface provides the options for sorting the information according to orientation and for selecting data by: item, instrument, measurement (function) or location.

The interface allows MOD to organize the materials data using custom filters and sorting procedures. The interface supports the use of pop-up menus that contain specific filters or sorting techniques that are appropriate to the certification or attestation requirement. Example operations are shown in

Laboratory Information Management System

Testing Sequence		Report Generation	
Fixed or Mobil	> <	Item Instrument Measurement	+ -
Filter: Instrument	=	Value: None	
Sort: None		Order: None	
Preview Report Available Saved Reports: Select Report		Save Report As: <input type="text"/> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team <input type="radio"/> RSS Standards <input type="radio"/> Gosstandards <input type="radio"/> QA/QC Determination Save Custom Report	

Figure 2.3b: Applying Instrument Filters. In this illustration, the MOD analyst is using an instrument filter to assess vendor specific data.

Laboratory Information Management System

Testing Sequence		Report Generation	
Fixed or Mobil	> <	Item Instrument Measurement	+ -
Filter: None	=	Value: None	
Sort: None		Order: Ascending	
Available Saved Reports: Select Report		Save Report As: <input type="text"/> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team <input type="radio"/> RSS Standards <input type="radio"/> Gosstandards <input type="radio"/> QA/QC Determination Save Custom Report	

Figure 2.3c: Applying Case Sorting Methods. In this illustration, the MOD analyst is instructing the database to organize all information in an ascending order with no additional filter procedure.

Figure 2.3b for the preview of selected filters by instrument class, and within Figure 2.3c for sorting query operations by item, fixed or mobile laboratory location, instrument type or measurement criteria.

Testing Sequence	Report Generation
	<div> <div>Item</div> <div>Instrument</div> <div>Measurement</div> <div>Fixed or Mobil</div> </div> <div> <div>+</div> <div>-</div> </div>
Filter: None	= Value: None
Sort: None	Order: Ascending
<div>Preview Report</div> <div>Available Saved Reports:</div> <div>Select Report</div>	<div>Save Report As:</div> <div></div> <div> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team </div> <div> <input type="radio"/> RSS Standards <input type="radio"/> Gosorteknadzor <input type="radio"/> QA/QC Determination </div> <div>Save Custom Report</div>

Item	Instrument	Measurement	Fixed or Mobil
1.1	ARC-MET930SP (Spectrum 18 OE Spectrometer)	Primary element detection and spectral analysis	Mobil
10.1	EFI 300 Leak Detector	Pressure detection-vacuum detection	Mobil
11.1	Testo 625 Hygrometer	Barometric pressure-hydrographic measurements	Mobil
12.1	NP-500 Hydraulic Press	Sample preparation and laboratory support	Mobil
13.1	Start 1M Compressor	Sample preparation and laboratory support	Mobil
14.1	HBM Dynamometer	Dynamic torque and acceleration	Mobil

Figure 2.3c: Accessing the Preview Report. In this illustration, the MOD analyst is searching the LIMS and the CTD Materials Database to examine the instruments in the fixed and mobile laboratory system. The query returns the information in a preview report that shows a small sub-section of the total database -- organized by item, instrument, measurement, and fixed or mobile location. The table has controls (pull-down menus) above each column for sorting each column according to a priori stated criteria. Since this application is written in open SQL and PL/SQL, the MOD can build custom filters to organize all data according to new and emerging criteria from RSS and Gosorteknadzor.

The preview report operation may be used to create a temporary view of the associated information. This operation creates a second pop-up window that allows the user to view a small sub-section of the data to determine the validity of the query process. In this manner, the user can quickly adjust the query procedure to define a second level operation or a refined search that conforms to the stated requirements. An example preview report is shown in Figure 2.3c. In this illustration, the user has selected all fields (item, instrument, measurement, and fixed or mobile location) to create a simple preview report. The report has the data organized in a tabular format that may be further sorted using the pull-down menus that are placed above each category.

Laboratory Information Management System

Testing Sequence		Report Generation	
Fixed or Mobil		<div>Item</div> <div>Instrument</div> <div>Measurement</div>	<div>+</div> <div>-</div>
Filter: None		Value: None	
Sort: None		Order: Ascending	
<div>Preview Report</div> <div>Available Saved Reports:</div> <div>Select Report</div>		<div>Save Report As:</div> <div></div> <div> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team </div> <div> <input type="radio"/> RSS Standards </div> <div> <input type="radio"/> Gosgiteknadzor </div> <div> <input type="radio"/> QA/QC Determination </div> <div>Save Custom Report</div>	

Item	Instrument	Measurement
1.1	ARC-MET930SP (Spectrum 18 CE Spectrometer)	Primary element detection and spectral analysis
10.1	EFI 300 Leak Detector	Pressure detection-vacuum detection
11.1	Testo 625 Hygrometer	Barometric pressure-hydrographic measurements
12.1	NP-600 Hydraulic Press	Sample preparation and laboratory support
13.1	Start 1M Compressor	Sample preparation and laboratory support
14.1	HBM Dynamometer	Dynamic torque and acceleration
15.1	Wika Pressure Transmitter	Pressure detection-vacuum detection

Figure 2.3d: Editing the Preview Report. In this illustration, the MOD analyst has removed the location column from the preview report. The query only shows three columns of information (item, instrument, measurement) without the fixed or mobile location. The data is presented in an ascending order with no additional filtering added into the database query. The pop-up window that displays the results, includes seven records (item 1.1 to item 15.1) with additional data that may be visualized by resizing the window. A series of scroll-bars are added to the pop-up report when the user wishes to visualize records that are outside the screen view for the dialog.

The addition and subtraction of data is illustrated in Figure 2.3d. In this example the location column has been removed from the database query. In addition, the analyst has resized the pop-up window to display additional data that includes a portion of the report for instrument 15.1: Wika Pressure Transmitter. As shown in the testing sequence view, all information from the fixed and mobile laboratory is being examined for this query search. At the present time, the query returns the information in an ascending order with no filters attached to restrict the search.

Item	Instrument	Measurement
5.1	A-Line 32D 12 Channel Acoustic Emission System	Network analysis using NDT-UT
1.1	ARC-MET930SP (Spectrum 18 OE Spectrometer)	Primary element detection and spectral analysis
17.1	Capillary Flow Equipment and Materials	Capillary and chemical amplification of surface features
F2.3	Charpy 300J Impact Tester	Primary hardness detection
18.1	Concrete Flaw Detector	NDT-UT (non-destructive testing, ultrasonic) flaw detection

Figure 2.3e: Selecting the CTD Instruments. In this illustration, the MOD analyst is beginning a new query based upon the HardTip 2000 instrument found within the fixed or mobile laboratory complex. The data shown in the preview report is now organized in an alphanumeric ascending order based upon an earlier query. For this reason, the preview shows the instruments beginning with A (A-Line 32D) and ending with C (Concrete Flaw Detector). The other instruments (ending with Z) are shown below this screen view.

The query is further refined using pop-up menus to identify the instrument type. This process is shown in Figure 2.3e. In this example, the user has selected the instrument pop-up menu to display all

working systems found within the fixed and mobile laboratory complex. The analyst has started a new query by selecting the HardTip 2000 instrument. Prior to this search, the database has oriented all instrumentation data in an ascending order. For this reason, the preview report displays a new set of equipment that is organized by vendor name in an alphanumeric ascending order (A-Line 12D to Concrete Flaw Detector with additional information displayed by resizing the preview window).

The query results for HardTip 2000 instrumentation are shown in Figure 2.3f. Since the CTD LIMS contains only one instrument from this vendor, a single record is provided in the preview report. Note that the user has also applied a request for metadata. Hence the preview report now shows a time/date stamp for the query that indicated when the report was requested. This information is then passed back to the main StarLIMS system for the final chain-of-custody report.

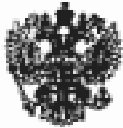
Laboratory Information Management System

Testing Sequence		Report Generation	
Fixed or Mobil		<div>Item</div> <div>Instrument</div> <div>Measurement</div> <div>+</div> <div>-</div>	
Filter: None	=	Value: None	
Sort: None		Order: Ascending	
<div>Preview Report</div> <div>Available Saved Reports:</div> <div>Select Report</div>		<div>Save Report As:</div> <div></div> <div> <input checked="" type="radio"/> Mobil Team <input type="radio"/> Fixed Team </div> <div> <input type="radio"/> RSS Standards <input type="radio"/> Gosgorteknadzor <input type="radio"/> QA/QC Determination </div> <div>Save Custom Report</div>	

Item	HardTip 2000 (Proceq)	Measurement
6.1	HardTip 2000 (Proceq Equotip Model D)	Surface hardness (pin destructive testing)
Report Generated on 11-MAR-2004 02:40:13		

Figure 2.3f: Database Query by Instrument Type. The illustration shows the result of the Hardtip 2000 query that was initiated on Figure 2.3e. Since the MOD facility uses only one instrument of this classification, a single record is displayed. The user can then query the database to retrieve all measurements and case studies (certifications, attestations, calibrations, etc.) that have been completed using this instrument.

The custom report is shown in Figure 2.3g. In this example, the user has selected information from the fixed laboratory facility and has removed all instrumentation that may be used for mobile operations. The query shows the instrument types organized by item F1.1 to F4.1 (additional records shown below this case example). As provided, the report indicates the main function for each instrument type and



12th Main Directorate

Laboratory Information Management System

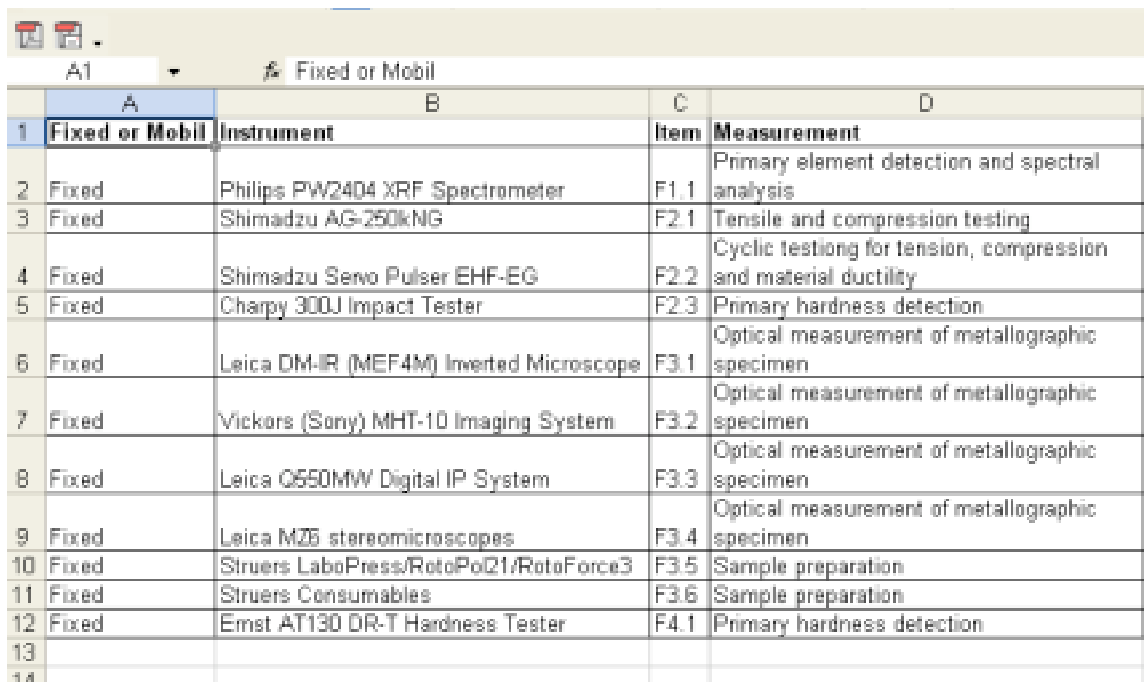
Download Report To MS Excel

Fixed or Mobil	Instrument	Item	Measurement
Fixed	Philips PW2404 XRF Spectrometer	F1.1	Primary element detection and spectral analysis
Fixed	Shimadzu AG-250kNG	F2.1	Tensile and compression testing
Fixed	Shimadzu Servo Pulser EHF-EG	F2.2	Cyclic testing for tension, compression and material ductility
Fixed	Charpy 300J Impact Tester	F2.3	Primary hardness detection
Fixed	Leica DM-IR (MEF4M) Inverted Microscope	F3.1	Optical measurement of metallographic specimen
Fixed	Vickors (Sony) MHT-10 Imaging System	F3.2	Optical measurement of metallographic specimen
Fixed	Leica Q550MW Digital IP System	F3.3	Optical measurement of metallographic specimen
Fixed	Leica MZ6 stereomicroscopes	F3.4	Optical measurement of metallographic specimen
Fixed	Struers LaboPress/RotoPol21/RotoForce3	F3.5	Sample preparation
Fixed	Struers Consumables	F3.6	Sample preparation
Fixed	Ernst AT130 DR-T Hardness Tester	F4.1	Primary hardness detection

Report Generated on 11-MAR-2004 02:48:41

Figure 2.3g: Custom Report Generation. In this illustration, the user has selected only case examples from the fixed laboratory system. The cases are organized by item identification code F1.1 to F4.1, with additional cases shown below this screen view. The report displays the measurement function for each instrument and includes the chain-of-custody metadata for when the report was requested by the MOD analyst. The migration tools for moving all data into Microsoft Excel are shown.

the time/date indicator for when this query report was generated. The report includes the fast migration tools for moving information into Microsoft Excel. This tool was requested by MOD to allow the Level I analysts to work with all information using standard spreadsheet functions. This migration tool is also used by the 12th Main Directorate to import/export data to the management teams located in Moscow. These analysts require summary reports and do not require specific tools that are localized to the scientific research facility in St. Petersburg.



	A	B	C	D
1	Fixed or Mobil	Instrument	Item	Measurement
2	Fixed	Philips PW2404 XRF Spectrometer	F1.1	Primary element detection and spectral analysis
3	Fixed	Shimadzu AG-250kNG	F2.1	Tensile and compression testing
4	Fixed	Shimadzu Servo Pulser EHF-EG	F2.2	Cyclic testing for tension, compression and material ductility
5	Fixed	Charpy 300J Impact Tester	F2.3	Primary hardness detection
6	Fixed	Leica DM-IR (MEF4M) Inverted Microscope	F3.1	Optical measurement of metallographic specimen
7	Fixed	Vickers (Sony) MHT-10 Imaging System	F3.2	Optical measurement of metallographic specimen
8	Fixed	Leica Q550MW Digital IP System	F3.3	Optical measurement of metallographic specimen
9	Fixed	Leica M25 stereomicroscopes	F3.4	Optical measurement of metallographic specimen
10	Fixed	Struers LaboPress/RotoPo21/RotoForce3	F3.5	Sample preparation
11	Fixed	Struers Consumables	F3.6	Sample preparation
12	Fixed	Ernst AT130 DR-T Hardness Tester	F4.1	Primary hardness detection
13				
14				

Figure 2.3h: Custom Reports in Microsoft Excel. In this illustration, the user is working with a LIMS custom report using the Excel migration tools. This provides a foundation for importing and exporting LIMS data into all Microsoft products that are readily used by the MOD field teams. The LIMS development includes tools for migrating data into Microsoft Word and Adobe PDF formats as well as most major graphical exchange formats (TIFF, GIF, JPEG).

The appearance of the custom report in Microsoft Excel is provided in Figure 2.3h. In this example the user has imported 12 records from the CTD LIMS query. The column titles and the orientation of the digital information are maintained throughout the process. As a component of this research project, we have provided a series of translation tools that allow MOD to move data across their specific applications. This includes graphical translators and report generation utilities. The tools for moving data are shown in the upper left corner of Figure 2.3h. In this illustration the Adobe PDF icon and the Adobe Interchange Icons are shown with a small pop-down arrow. This allows the MOD analysts to move data across vendors in an independent manner for mobile operations on portable notebook systems.

3.0 Algorithm Development and Structured Database Management

Within this discussion, we examine the algorithms and database structures that work with the 12th Main Directorate LIMS. The enclosed procedures have been requested by MOD to assist their engineering staff in the development of stand-alone processes for the management of materials data. This includes technical information that is formally stored within the LIMS system, and ancillary data that is stored within the Oracle RDBMS (for materials management). The algorithms are developed within the procedural language SQL and the Oracle proprietary Procedural Language PL/SQL. In each case, the algorithms operate independently from the formal LIMS structure – however, *all* algorithms may be added to the LIMS process since the StarLIMS operates using SQL and PL/SQL standards from both Sybase and Oracle.

The variations in SQL and PL/SQL may be summarized as follows:

- a. Procedural Language SQL (PL/SQL) is the Oracle proprietary procedural language extension to the industry-standard Structured Query Language (SQL). PL/SQL emphasizes data abstraction, information hiding, and other key elements of modern design strategies.
- b. The language PL/SQL incorporates: a full range of data types, explicit block structures, conditional and sequential control statements, loops of various modes and structures, exception handlers for use in event-based error modeling, constructs that help in the development of modular code-functions, procedures, and packages (collections of related programs and variables).

PL/SQL is also a block-oriented language. Hence, all code is organized into one or more blocks demarked by BEGIN and END statements. These blocks provide a high degree of structure, making it easier to develop and maintain code. This feature is particularly important to the MOD 12th Main Directorate, since data and processes may be organized according to technical requirements, and a common procedure library may be used by all technical teams (mobile and fixed laboratories). The algorithms may be installed at the desktop and mobile field laboratory level in a common-laptop environment.

3.1 PL/SQL Organization and Structure

PL/SQL functions and procedures, including packaged procedures and anonymous blocks, encapsulate a sequence of statements and the following basic layout:

Structure:

Header: Relevant for named blocks only. The header determines the way the named block or program is called. An example is shown in Figure 3.1a with the main sections that are generally included in a PL/SQL algorithm, function, or procedure.

When called by client applications (or the StarLIMS main shell), the PL/SQL procedure can accept arguments, reference other procedures or functions (or otherwise return values to a secondary process). Compiled in an executable form, procedure calls are quick and efficient. Executable code is automatically cached and shared among users as a method to lower memory requirements, and invocation overhead. By grouping SQL statements, a stored procedure allows an organized structure (of process steps, functions, sub-tasks, and ancillary procedures) to be executed within a single

reference call.² An example of this process is shown in Figure 3.1b for a priori stored procedures, and Figure 3.1c for package-stored formats. In each example, a simple test string is used to illustrate the organization and structure of the reference process.

```
[PROCEDURE name IS] or [FUNCTION name RETURN datatype IS]

DECLARE
  /* declarations */
  The part of the block that declares variables, cursors,
  and sub-blocks that are referenced in the execution and exception sections.

BEGIN
  /* executable code */
  The part of the PL/SQL block containing the executable statements,
  the code that is executed by the PL/SQL runtime engine

  [RETURN value]          <-- for functions

EXCEPTION
  /* error handling */
  The section that handles exceptions to normal processing
  (warnings and error conditions)

END;
```

Or a flat file SQL script can contain simply:

```
BEGIN
  /* executable code */

EXCEPTION
  /* error handling */

END;
```

Figure 3.1a: PL/SQL Standard Structure with Declarations, Body, and Exception Handling.

SQL is a *set-at-a-time* database language. Hence, the user cannot selectively examine, or modify a single row from a SELECT statement's result set. By extension, PL/SQL gives the user the ability to handle data one row at a time. As the user executes the SQL statement from within PL/SQL, a private work area is assigned for that statement. This is analogous to the methods that are used within StarLIMS, where the set-aside user environment is used to manage the basic data input-output, as well as, the organizational structure of the data. The partition also controls the security and process management (including chain-of-custody considerations). The private work area is managed at the systems administration level. This provides the strict control that is required from the 12th Main Directorate (security and alarm provisions) with the flexibility that is required to add and subtract basic processes.

That partition area contains the information about the SQL statement and the set of data returned by the statement. Within SQL, the cursor is a mechanism by which you can name that work area and

² This minimizes the use of slow networks, reduces network traffic, and improves round-trip response time. Additionally, stored procedures enable the user to take advantage of computing resources on the ML-530 server. For example, you can move computation-bound procedures from client to server, where they will execute in a highly efficient manner. Likewise, stored functions called from SQL statements enhance performance by executing application logic within the server or partitioned disk array.

manipulate the information within it. Explicit Cursor is a SELECT statement that is explicitly defined in the declaration section and assigned a name. This gives the analyst complete control over how to access information subject to their respective abilities (certification levels) to access the database structure or assigned function (Figure 3.1d).

```

Create or Replace PROCEDURE PROCEDURE_DEMO
IS
demoString Varchar2(15):= 'THIS IS A TEST';
BEGIN
dbms_output.put_line(demoString);
END PROCEDURE_DEMO;

SQL> set serveroutput on;
SQL> begin
  2  procedure_demo;
  3  end;
  4  /

or

exec procedure_demo;

THIS IS A TEST
PL/SQL procedure successfully completed.

```

Figure 3.1b: PL/SQL Sample Stored Procedure Format. This simple procedure creates the test string *demoString* that is used to demonstrate the structure of a stored PL/SQL procedure. The *Varchar2* data type stores the string and the *dbms_output* function displays the variable when the procedure is executed.

```

Create or Replace PACKAGE PACKAGE_DEMO AS
  PROCEDURE PACKAGE_DEMO_PROCEDURE;
END PACKAGE_DEMO;

CREATE OR REPLACE PACKAGE BODY PACKAGE_DEMO AS
  PROCEDURE PACKAGE_DEMO_PROCEDURE
  IS
demoString Varchar2(15):= 'THIS IS A TEST';
BEGIN
dbms_output.put_line(demoString);
END PACKAGE_DEMO_PROCEDURE;
END PACKAGE_DEMO;

SQL> BEGIN
  2  PACKAGE_DEMO.PACKAGE_DEMO_PROCEDURE;
  3  END;
  4  /
THIS IS A TEST

Or exec Package_demo.package_demo_procedure;

PL/SQL procedure successfully completed.

```

Figure 3.1c: PL/SQL Sample Stored Package Format. This simple procedure creates a test string that is used to demonstrate the structure of a stored PL/SQL procedure created as a package. The package format enables the procedures to be grouped together as related elements. The *Varchar2* data type stores the string and the *dbms_output* function displays the variable.

```

Create or Replace FUNCTION FUNCTION_DEMO Return Varchar2
IS
demoString Varchar2(15):= 'THIS IS A TEST';
BEGIN
    RETURN demoString;
END FUNCTION_DEMO;

SQL> SELECT FUNCTION_DEMO() FROM DUAL;
FUNCTION_DEMO()

THIS IS A TEST

```

Figure 3.1d: PL/SQL Sample Stored Function Format. This simple function creates a test string that is used to demonstrate the structure of a stored PL/SQL procedure created as a package. The *Varchar2* data type stores the string and the *dbms_output* function displays the variable.

3.2 Cursors and Database Interaction

Cursors are divided into two groups, implicit and explicit. An explicit cursor is a *select statement* that is explicitly defined in the declaration of a PL/SQL block. Once created it is then processed with OPEN, FETCH and CLOSE statements. Using implicit cursors, the RDBMS opens, fetches and closes the cursor automatically. Although implicit cursors are (in many instances) faster than explicit cursors, they are vulnerable to data errors, and allow less programmatic control within the networked environment. The re-application of an implicit cursor is only possible by recalling the procedure that initialized it. Conversely, the explicit cursor can be reused (recycled). This process increases the chance that it will be pre-parsed in shared memory when needed - significantly increasing the performance of the query. In all cases, the explicit cursor utilizes the *dot notation* to reference fields within the cursor. A simple example for this configuration is shown in Figure 3.2a. By stepping through the record set the user can examine every record returned by the cursor query.

```

DECLARE
MACHNAME VARCHAR2(100);
BEGIN
SELECT MACHINE_NAME
INTO MACHNAME
FROM TEST2_TBL
WHERE MACHINE_ID=1;
    dbms_output.put_line('Result is:  ' || MACHNAME);
END;
/

Result is:  Machine 1

```

Figure 3.2a: Implicit Cursor. Within this example, an implicit cursor is used to select data from a SQL statement into the variable *MACHNAME*. An error will result if the query returns no data or more than a single value. This forces the user to use error handling and trap for conditions that will result in errors.

In Figure 3.2b-g, the use and application for the explicit cursor is identified using numerous case examples. In Figure 3.2b-c, the dynamics for the basic loop procedure are identified using the standard loop configuration and the while-loop structure. The illustration is extended in Figure 3.2d-g using explicit cursors to demonstrate certain % functions such as %NOTFOUND, %ISOPEN, and %ROWCOUNT. Each function and procedure is used by MOD to parse materials data according to

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
      FROM test_tbl;
  the_result exp_cursor%ROWTYPE;
BEGIN
  OPEN exp_cursor;
  // Format Headings
  dbms_output.put_line('Report Heading');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Column One',20,' ')||' '||
    rpad('Column Two',30,' '));
  dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));
  //loop through cursor
  LOOP
    FETCH exp_cursor INTO the_result;
    EXIT WHEN exp_cursor%NOTFOUND;
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
  END LOOP;
  CLOSE exp_cursor;
END;
/

```

Figure 3.2b: Explicit General Cursor. In this example, the cursor function is used to demonstrate a simple cursor. A small loop is created to place the results of the cursor query into the variable *the_result*. That variable is of the datatype %ROWTYPE which provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable. The loop will terminate when the *exp_cursor* contains no data (%NOTFOUND).

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
      FROM test_tbl;
  the_result exp_cursor%ROWTYPE;
BEGIN
  OPEN exp_cursor;
  //Format Headings
  dbms_output.put_line('Report Heading');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Column One',20,' ')||' '||
    rpad('Column Two',30,' '));
  dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));
  FETCH exp_cursor INTO the_result;
  //Loop Through Cursor
  WHILE (exp_cursor%FOUND) LOOP
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
    FETCH exp_cursor INTO the_result;
  END LOOP;
  CLOSE exp_cursor;
END;
/

```

Figure 3.2c: Explicit Cursor Using a While Loop. This is the same as Figure 3.2b except that the cursor is accessed using a while loop. Instead of the %NOTFOUND cursor attribute used previously, as an alternative %FOUND attribute continues the LOOP as long as data is present.

storage identifiers. Tabular data is extracted using either column or row delineation methods. Whereas, the logical not-found and is-open process steps are used to examine the availability of the data structure or materials data volume.

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
    FROM test_tbl;
    the_result exp_cursor%ROWTYPE;
BEGIN
  IF (NOT exp_cursor%ISOPEN) THEN
    OPEN exp_cursor;
  END IF;
  /*Format headings */
  dbms_output.put_line('Report Heading');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Column One',20,' ')||' '||
    rpad('Column Two',30,' '));
  dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));
  FETCH exp_cursor INTO the_result;
  WHILE (exp_cursor%FOUND) LOOP
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
    FETCH exp_cursor INTO the_result;
  END LOOP;
  IF(exp_cursor%ISOPEN)THEN
    CLOSE exp_cursor;
  END IF;
END;
/

```

Figure 3.2d: Explicit Cursor Using %ISOPEN. In this example, the cursor is tested to make sure it is not already open before attempting to open it and that it is not closed before attempting to close it. Attempting to close a cursor that is not open will produce an ORA-01001: Invalid cursor error. Attempting to open a cursor that is already open will produce an ORA-06511 "Cursor Already Open" error.

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
    FROM test_tbl;
    num_total_rows NUMBER;
BEGIN
  /*Format headings */
  dbms_output.put_line('Report Heading');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Column One',20,' ')||' '||
    rpad('Column Two',30,' '));
  dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));
  FOR the_result IN exp_cursor LOOP
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
    num_total_rows :=exp_cursor%ROWCOUNT;
  END LOOP;
  IF num_total_rows >0 THEN
    dbms_output.new_line;
    dbms_output.put_line('Report Completed...Total Rows:'||num_total_rows);
  END IF;
END;
/

```

Figure 3.2e: Explicit Cursor Using %ROWCOUNT. . This example demonstrates the %ROWCOUNT attribute of the cursor. The variable `num_total_rows` accumulates the number of records returned by the cursor. After the LOOP terminates, the total number of records retrieved is displayed.

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
    FROM test_tbl;
  num_total_rows NUMBER;
BEGIN
  FOR the_result IN exp_cursor LOOP
    IF exp_cursor%ROWCOUNT =1 THEN
      /*Format headings */
      dbms_output.put_line('Report Heading');
      dbms_output.put_line('-----');
      dbms_output.put_line(rpad('Column One',20,' ')||' '||
        rpad('Column Two',30,' '));

      dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));
    END IF;
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
    num_total_rows :=exp_cursor%ROWCOUNT;
    dbms_output.put_line(num_total_rows);
  END LOOP;
END;
/

```

Figure 3.2f: Explicit Cursor Using %ROWCOUNT as an Incremental Rowcount. This example illustrates incrementing the %ROWCOUNT attribute as the LOOP is executed. For each pass through the LOOP, the variable `num_total_rows` is incremented and displayed.

```

DECLARE
  CURSOR exp_cursor IS
    SELECT testvalue1,testvalue2,testvalue3
    FROM test_tbl;
BEGIN
  /*Format headings */
  dbms_output.put_line('Report Heading');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Column One',20,' ')||' '||
    rpad('Column Two',30,' '));
  dbms_output.put_line(rpad('-',20,'-')||' '||rpad('-',30,'-'));

  For the_result in exp_cursor LOOP
    dbms_output.put_line(rpad(the_result.testvalue1,20,' ')||' '||
      rpad(the_result.testvalue2,30,' '));
  END LOOP;
//Note...It is not necessary to close the cursor
END;
/

```

Figure 3.2g: Explicit Cursor Using a Cursor For Loop. This type of cursor reduces the volume of code needed to fetch data. It also greatly lessens the chance of introducing loop errors in the programming. The loop terminates unconditionally when all of the records in the associated cursor have been fetched. It eliminates the declaration of the record, the OPEN, FETCH and CLOSE statements and the %FOUND or %NOTFOUND attributes. Although there are many advantages to this cursor, it is not appropriate when you need to apply conditions to each fetched record to determine if you should halt the loop. Although it is possible to terminate the LOOP with an EXIT statement, it is not recommended.

3.3 Adding Parameters to the Cursors

Parameters can be added to the cursors to give them the ability to pass information to the SQL statement. In this manner the cursor may be reused for various tasks within the main materials database. The addition of parameters also avoids scoping problems, since the cursor is not tied to a specific variable in a program or block or a specific logic statement. If the procedure has nested blocks, the cursor can be defined at a higher-level and then used in any of the sub-blocks with variables defined within that block.

An example process is shown in Figure 3.3a using WHERE clause to select particular information. In this case study, the parameter *machine_id* is passed to the cursor when *exp_cursor* is opened. The FETCH statement is then used to place the *exp_cursor* into the resultant query.

```

DECLARE
//id is parameter that will be passed to SELECT statement
CURSOR exp_cursor(id number) IS
  SELECT machine_id,machine_name,machine_cost
  FROM test2 tbl
  WHERE machine_id=id;
  the_result exp_cursor%ROWTYPE;
BEGIN
  //open the cursor and pass the value of 1 to the SELECT statement
  OPEN exp_cursor(1);
  /*Format headings */
  dbms_output.put_line('Machine Data');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
  rpad('Machine Name',30,' ')||' '||
  rpad('Machine Cost',30,' '));
  dbms_output.put_line('-----');
  LOOP
    FETCH exp_cursor INTO the_result;
    EXIT WHEN exp_cursor%NOTFOUND;
    dbms_output.put_line(rpad(the_result.machine_id,20,' ')||' '||
    rpad(the_result.machine_name,30,' ')||' '||
    rpad(the_result.machine_cost,30,' '));
  END LOOP;
  CLOSE exp_cursor;
END;
/

```

Figure 3.3a: Explicit Cursor Using a Parameter. The parameter *machine_id* is passed to the cursor when *exp_cursor* is opened. The value is then inserted into the WHERE clause of the cursor when the statement is executed.

The algorithm shown in Figure 3.3a, creates an output of the following format:

Machine Data		
Machine ID	Machine Name	Machine Cost
1	Machine 1	150000

In this example, the machine cost (service cost) for the first instrument (Machine ID = 1) is provided. As shown, the formatting for the result uses standard ASCII characters and may be explicitly edited by the end user based upon reporting requirements and table standards. A second example is shown in Figure 3.3b for the explicit cursor using a parameter. However, this case study uses the FOR LOOP

structure to create the tabular output. This example also illustrates the use of the *dbms_output_line* procedure for creating standardized (line-oriented) outputs at a target location. The *rpad* function is nested within the *dbms_output_line* to create the required (line-oriented) spacing that is required for the table indentations.

```

DECLARE
//id is parameter that will be passed to SELECT statement
CURSOR exp_cursor(id number) IS
    SELECT machine_id,machine_name,machine_cost
    FROM test2_tbl
    WHERE machine_id=id;
BEGIN
    /*Format headings */
    dbms_output.put_line('Machine Data');
    dbms_output.put_line('-----');
    dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
    rpad('Machine Name',30,' ')||' '||
    rpad('Machine Cost',30,' '));
    dbms_output.put_line('-----');
    //open the cursor and pass the value of 1 to the SELECT statement
    for the_result in exp_cursor(1) LOOP
        dbms_output.put_line(rpad(the_result.machine_id,20,' ')||' '||
        rpad(the_result.machine_name,30,' ')||' '||
        rpad(the_result.machine_cost,30,' '));
    END LOOP;
END;
/

```

Figure 3.3b: For Loop Cursor Using a Parameter. The format is the same as the previous example except the parameter is passed in the LOOP.

3.4 Adding Bind Variables To The Cursor

The CTD Materials Database has multiple execution engines including one each for PL/SQL and SQL. When running PL/SQL blocks and subprograms, the PL/SQL engine runs all procedural statements and sends the SQL statements to the SQL engine. Within the SQL engine, the statements are parsed and executed using ASCII standards. The results from the SQL engine are then passed back to the PL/SQL engine for interpretation. During execution, every SQL statement causes a context switch between the two engines, which results in a performance penalty. This is the basic handshaking that is required to support Oracle standards for PL/SQL and blind (external) standards for ASCII SQL.

As demonstrated in the StarLIMS, the performance can be substantially improved by minimizing the number of context switches required to run a particular block or subprogram. For example, the standard functions for graphical user interfacing, data I/O, chain-of-custody, and report generation can be performed without the need to submit information into PL/SQL provided the materials data is local within the laboratory network. Conversely, the MOD analyst that utilizes archived information from the Oracle ML-530 Server uses PL/SQL to access this information and supplement the local data that is required for the testing sequence.

When a cursor is executed frequently, the same exact SQL statement will be submitted repeatedly causing the database to parse and compile each submission. This is referred to as a *hard parse*. Bind variables, signified by preceding a placeholder with ":", become beneficial when it is discovered that

the network is using multiple copies of the same query that only differ by values.³ Under this condition, the judicious use of bind variables will increase performance, and lessen the burden on the database server. A simple example for the bind variable is shown in Figure (3.4a). In this example, the placeholder identification is directly inserted into the select statement. When the procedure is executed, the identification (:ID) is inserted into the SELECT statement.

```

DECLARE
stmt varchar2(1000);
type mbrCursorType is ref cursor;
iCursor mbrCursorType;
machine_id number;
machine_name varchar2(100);
machine_cost number;
BEGIN
Stmt:='SELECT MACHINE_ID,MACHINE_NAME,MACHINE_COST FROM TEST2_TBL
      WHERE MACHINE_ID=:ID';
OPEN iCursor FOR stmt using 1;
Fetch iCursor into machine_id,machine_name,machine_cost;
  /*Format headings */
  dbms_output.put_line('Machine Data');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
rpad('Machine Name',30,' ')||' '||
rpad('Machine Cost',30,' '));
  dbms_output.put_line('-----');
LOOP
  FETCH iCursor into machine_id,machine_name,machine_cost;
Exit when iCursor %NOTFOUND;
  dbms_output.put_line(rpad(machine_id,20,' ')||' '||
rpad(machine_name,30,' ')||' '||
rpad(machine_cost,30,' '));
END LOOP;
END;
/

```

Figure 3.4a: Explicit Cursor With Bind Variables. A placeholder :ID is inserted into the SELECT statement when the procedure is compiled. At runtime, :ID will be replaced with the value specified in the USING clause of the OPEN statement.

In the design of the CTD Materials database, *bulk* binding is used to improve performance by reducing the number of context switches required to run SQL statements. Bulk binding is binding an entire collection at once rather than iteratively. With bulk binding, entire collections, not just individual elements are passed back and forth between PL/SQL and SQL. Hence, the SQL engine will load all the values of the columns into nested tables before returning them to PL/SQL engine, so there will be only one context switch no matter how many rows are returned. Bulk binds are most efficient for:

- SQL statements within PL/SQL loops,
- Cases that require a collection of elements efficiently organized within a binding,
- Cases that require four or more rows together in iteration. The more rows affected by a SQL statement, the greater the procedural efficiency.

³ The variables are called bind variables since the values that are passed to the SQL statement are bound to the query at runtime. This is also known as a positional bind since the placeholders are referred to by their position in the statement rather than their names.

An example bulk collection is shown in Figure 3.4b. In this procedure, the data selection is inserted into the variables shown in the *FETCH* statement. Note that data location varies by array position and the entire procedure is dynamic with respect to the location of the original data SQL or PL/SQL and the position of the information within the respective RDBMS (Oracle, Sybase, or local LIMS structure).

```

DECLARE
    stmt          varchar2(1000);
    type mbrCursorType is ref cursor;
    iCursor mbrCursorType;
    type test_type is table of varchar2(4000);
    machine_id     test_type;
    machine_name   test_type;
    machine_cost   test_type;
BEGIN

    Stmt:='SELECT MACHINE_ID,MACHINE_NAME,MACHINE_COST FROM TEST2_TBL
           WHERE MACHINE_ID=:ID';
    OPEN iCursor FOR stmt using 1;

    FETCH iCursor bulk collect into machine_id,machine_name,machine_cost;

    /*Format headings */
    dbms_output.put_line('Machine Data');
    dbms_output.put_line('-----');
    dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
    rpad('Machine Name',30,' ')||' '||
    rpad('Machine Cost',30,' '));
    dbms_output.put_line('-----');

    for I in MACHINE_ID.FIRST..MACHINE_ID.LAST LOOP
        dbms_output.put_line(rpad(machine_id(i),20,' ')||' '||
        rpad(machine_name(i),30,' ')||' '||
        rpad(machine_cost(i),30,' '));
    END LOOP;
END;
/

```

Figure 3.4b: Explicit Cursor Bulk Collect With Bind Variable. The entire SELECT statement is inserted into the variables specified in the FETCH statement in an array-like format. The data is extracted from the variables by specifying the location within the array.

3.5 Applications for Dynamic SQL and DBMS_SQL

Dynamic SQL and DBMS_SQL allow the user to execute process statements that are not parsed or bound during the compile sequence. Hence, the statement is constructed at runtime, and then is passed to the SQL engine for processing. The dynamic methods are required when exact information concerning the database structure (or tabular data) is not known prior to the compiling sequence. Within the CTD network, this occurs when two or more users are accessing and modifying data from the same source without notification from the systems administrator. This condition also occurs when mobile teams are adding and subtracting information that is placed within the central RDBMS server. Hence the analyst may be accessing information that has significantly changed from the prior processing cycle.

Dynamic SQL programs can handle changes in data definitions, without the need to recompile the entire code resource. This makes dynamic SQL much more flexible than static SQL. In addition, dynamic SQL lets the analyst write reusable code since the statements are quickly adapted for varying

environments. An example algorithm for dynamic SQL is shown in Figure (3.5a). In this case example, the parameter *sqlstmt* is shown at the top of the process and uses a dynamic variable to pass information into the create (or replace) procedure. As indicated, the algorithm generates a formatted table with Rockwell hardness values acquired from MOD sample sections. The table shows only two of the observed sample values, however, the full sequence would include all values and observations from the RDBMS.

```
CREATE OR REPLACE procedure dbms1_demo
(sqlstmt in varchar)
IS
  MACHINE_ID VARCHAR2 (50);
  MACHINE_NAME VARCHAR2 (50);
  MACHINE_COST VARCHAR2 (50);
  type mbrCursorType is ref cursor;
  mbrCursor mbrCursorType;
BEGIN
  // Format Report Headings
  dbms_output.put_line('Machine Data');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
    rpad('Machine Name',30,' ')||' '||
    rpad('Rockwell Hardness',30,' '));
  dbms_output.put_line('-----');
  OPEN mbrCursor for sqlstmt;

  LOOP
    FETCH mbrCursor INTO machine_id,machine_name,sample_result;
    EXIT WHEN mbrCursor%NOTFOUND;
    dbms_output.put_line(rpad(machine_id,20,' ')||' '||
      rpad(machine_name,30,' ')||' '||
      rpad(sample_result,30,' '));
  END LOOP;
end;
/
```

Figure 3.5a: Simple Dynamic SQL Structure. In this example, the SQL statement is passed to the procedure through the parameter *sqlstmt*. This method offers improved flexibility, since the data that will be selected does not have to be known before runtime. The algorithm produces an output of the following format:

```
SQL> exec dbms1_demo('select * from test2_tbl')
Machine Data
-----
Machine ID          Machine Name          Rockwell Hardness
-----
1                   Machine 1              150000
2                   Machine 2              160000
```

In this output sequence, the user has executed the SQL procedure *dbms1_demo* using the select parameters from the *test2_tbl* database. To simplify this case example, only two observations are shown. Each observation indicates the machine ID, name and respective measurement (Rockwell Hardness value).

A second dynamic SQL sequence is shown in Figure (3.5b). In this example, the algorithm is used to determine the number of rows in a database table. This procedure uses the dynamic variable *p_tname* to pass information into the main process body, and returns the dynamic size of the database with respect to the number of observations in the row or column format. The algorithm is particularly useful for case conditions that utilize rows as observations and columns as variable definitions. This is

generally the condition for all RDBMS constructions and is always the case for the analysis of materials data within a standard statistical package such as SAS, SPSS, or their equivalent.

```

create or replace
function get_rows( p_tname in varchar2 )          return number
is
    l_theCursor    integer default dbms_sql.open_cursor;
    l_columnValue  number default NULL;
    l_status       integer;
begin
    dbms_sql.parse( l_theCursor,
                    'select count(*) from ' || p_tname,
                    dbms_sql.native );

    dbms_sql.define_column( l_theCursor, 1, l_columnValue );
    l_status := dbms_sql.execute(l_theCursor);

    if ( dbms_sql.fetch_rows(l_theCursor) > 0 )
    then
        dbms_sql.column_value(l_theCursor, 1, l_columnValue);
    end if;
    dbms_sql.close_cursor( l_theCursor );
    return l_columnValue;
end ;
/

```

Figure 3.5b: Dynamic SQL Example – Row Count Identification. This function is used to generate the number of rows in any table passed as a parameter into the main body. The function is also required for determining LOOP sizes for unknown conditions and may be used as an output tool for formatting reports and header tables. The algorithm produces an output of the following format:

```

SQL> select get_rows('test2_tbl') from dual;

GET_ROWS('TEST2_TBL')
-----
5

```

In this output sequence, the user has executed the SQL procedure *get_rows* using the select parameters from the *test2_tbl* database. The simple example shows an output of five rows (observations) from the RDBMS. The row example may be further modified to extract the number of columns from within the table construction. This modification is shown within Figure 3.5c, where the numbers of columns from the RDBMS are extracted (as a parameter), that may be used as an input declaration within a second SQL process.⁴

For the development of the CTD LIMS, the dynamic SQL methods are used during data acquisition to extract information from the respective instruments in the fixed and mobile laboratories. The information is acquired using the LOOP or WHILE procedures to test the observations for range and error detection (the initial QA/QC process) as well as to parse the information according to line orientation. This includes the parsing of information using simple column, space, comma, and tab delimiters that are used to separate valid observations. The LIMS also checks end-of-record delimiters and end-of-file delimiters as a means to identify the availability of the technical information. The use of dynamic procedures to check for the size of the database construction (rows by columns) is useful

⁴ This function is also required for statistical identification of the number of variables and covariates that are found within a case study.

for comparing standardized testing sequences such as repeated measures. For example, the use of this technique for acquiring information from the Shimadzu system is important for repeated linear test and servo-pulser examinations. For the servo-pulser, repeated measures are used to simulate a sequence of tensile and compressive tests that occur under field conditions such as repeated jarring or bending of the sample under difficult load-bearing environments.

```
CREATE OR REPLACE procedure getcolumns
(tablename in varchar)
IS
COLUMNNAME VARCHAR2(50);
SQLSTMT VARCHAR2(100);
type mbrCursorType is ref cursor;
mbrCursor mbrCursorType;
BEGIN
  SQLSTMT:='select COLUMN_NAME from user_tab_columns where table_name= '''||TABLENAME||''';
  DBMS_OUTPUT.PUT_LINE(sqlstmt);
  OPEN mbrCursor for sqlstmt;

  LOOP
    FETCH mbrCursor INTO COLUMNNAME;
    EXIT WHEN mbrCursor%NOTFOUND;
    dbms_output.put_line(COLUMNNAME);
  END LOOP;
end;
/
```

Figure 3.5c: Dynamic SQL Example – Column Identification. This procedure will return the column names for any table specified as an input parameter. The algorithm may be used in conjunction with the example shown in Figure 3.5b to produce a report with column headings and data definitions. The algorithm produces an output of the following format:

```
SQL> exec getcolumns('TEST2_TBL');

MACHINE_ID
MACHINE_NAME
MACHINE_MEASURE
```

In this output sequence, the user has executed the SQL procedure *getcolumns* using the select selected *tablename* query. To simplify this case example, only three column headers are shown (*Machine_ID*, *Machine_Name*, and *Machine_Measure*).

In Figure 3.5d, a partially dynamic process is shown. In this case example, the variable sizes (array sizes) are declared a priori however the ORDERBY clause is used as a method to dynamically alter the results according to stated criteria. The algorithm creates a tabular output similar to Figure (3.5a), however, the results also show how a new instrument (new machine) may be identified and added into the database.⁵

⁵ The ORDERBY process is particularly useful for sorting cases and observations within the database. The ordering methods have been included within the StarLIMS software to allow MOD to sort cases, observations, and testing sequences as required to meet 12th Main Directorate requirements. The use of ORDERBY methods was required for the generation of reports from within the main CTD materials database. This process is used independently from the LIMS, and is commonly required by MOD to generate sub-reports for internal distribution across the CTD laboratory system.

```

declare
MACHINE_ID VARCHAR2 (50);
MACHINE_NAME VARCHAR2 (50);
MACHINE_COST VARCHAR2 (50);
type mbrCursorType is ref cursor;
mbrCursor mbrCursorType;
sqlstmt varchar(100):='select * from test2_tbl';
orderby varchar(25):='Machine_name';

BEGIN
  if orderby is not null then
    sqlstmt:=sqlstmt||' order by '||orderby;
  end if;

  dbms_output.put_line('Machine Data');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
rpad('Machine Name',30,' ')||' '||
rpad('Machine Cost',30,' '));
  dbms_output.put_line('-----');
  OPEN mbrCursor for sqlstmt;

  LOOP
    FETCH mbrCursor INTO machine_id,machine_name,machine_cost;
    EXIT WHEN mbrCursor%NOTFOUND;
    dbms_output.put_line(rpad(machine_id,20,' ')||' '||
rpad(machine_name,30,' ')||' '||
rpad(machine_cost,30,' '));
  END LOOP;
end;
/

```

Figure 3.5d: Appending Information into an SQL Statement. In this example, one component of the SQL statement is known, and is not expected to change after it is compiled. The variable portion of the statement (the ORDERBY clause) is passed to the algorithm in a dynamic manner as the procedure executes. The algorithm produces an output of the following format:

```

Machine Data
-----
Machine ID      Machine Name      Rockwell Hardness
-----
1               Machine 1          150000
2               Machine 2          160000
3               Machine 3          170000
0               New Machine       250000

```

PL/SQL procedure successfully completed.

In this output sequence, four cases are shown for Rockwell hardness testing from within the RDBMS. A new machine has been added into the database and the cases are sorted in ascending order with the new machine added at the end of the record.

In Figure 3.5e, the prior examples are extended to include methods for dynamic reporting of parameters, cases, and labels. This procedure is used by MOD as a base-line example for the creation of customized reports. In this example, the SQL statements are contained within the RDBMS, and the final output includes the query data with labels and context information.

```

CREATE TABLE .SQL_TBL
(
  SQL_NAME VARCHAR2(50),
  SQL_STMT VARCHAR2(500)
)

INSERT INTO SQL_TBL VALUES('SQLSTMT1','SELECT MACHINE_ID,MACHINE_NAME,MACHINE_MEAS FROM
TEST2_TBL WHERE MACHINE_ID=1');

INSERT INTO SQL_TBL VALUES('SQLSTMT2','SELECT MACHINE_ID,MACHINE_NAME,MACHINE_MEAS FROM
TEST2_TBL WHERE MACHINE_ID=2');

COMMIT;

CREATE OR REPLACE procedure dbms1_demo
(STMTNAME in varchar)
IS
SQLSTMT VARCHAR2(1000);

type mbrCursorType is ref cursor;
mbrCursor mbrCursorType;
type gCursorType is ref cursor;
gCursor gCursorType;
machine_id number;
machine_name varchar2(100);
machine_cost number;

BEGIN
  sqlstmt:='SELECT SQL_STMT FROM SQL_TBL WHERE SQL_NAME=:THENAME';
                                open gCursor for sqlstmt using STMTNAME;
                                fetch gCursor into SQLSTMT;
                                close gCursor;

  dbms_output.put_line('Rockwell Hardness');
  dbms_output.put_line('-----');
  dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
rpad('Machine Name',30,' ')||' '||
rpad('Machine Cost',30,' '));
  dbms_output.put_line('-----');

  DBMS_OUTPUT.PUT_LINE(sqlstmt);
  OPEN mbrCursor for sqlstmt;

  LOOP
    FETCH mbrCursor INTO machine_id,machine_name,machine_meas;
    EXIT WHEN mbrCursor%NOTFOUND;
    dbms_output.put_line(rpad(machine_id,20,' ')||' '||
rpad(machine_name,30,' ')||' '||
rpad(machine_cost,30,' '));
  END LOOP;

  CLOSE mbrCursor;
end;
/

```

Figure 3.5e: Retrieve SQL Statement From Database Table. In this example a SQL statement is stored in a database table. By selecting the record that contains the SQL statement, a dynamic report is generated and displayed to the user. This procedure identifies the methods that are required to build report labels and report queries from StarLIMS and from the external CTD materials database. The methods use independent SQL and PL/SQL techniques to access this information and may be independently used by the field (mobile) teams working outside the StarLIMS environment. The algorithm produces an output of the following format:


```
SQL> exec dbms1_demo('SQLSTMT1')
Machine Data
-----
Machine ID          Machine Name          Rockwell Hardness
-----
SELECT MACHINE_ID,MACHINE_NAME,MACHINE_COST FROM TEST2_TBL WHERE MACHINE_ID=1
1              Machine 1              150000

PL/SQL procedure successfully completed.

SQL> EXEC dbms1_demo('SQLSTMT2')
Machine Data
-----
Machine ID          Machine Name          Rockwell Hardness
-----
SELECT MACHINE_ID,MACHINE_NAME,MACHINE_MEAS FROM TEST2_TBL WHERE MACHINE_ID=2
2              Machine 2              160000

PL/SQL procedure successfully completed.
```

In this output sequence, two queries are shown for Rockwell hardness testing from within the RDBMS. Each query acts on the procedure *dbms1_demo*, however the results depend upon the inserted SQL statement *SQLSTMT1* or *SQLSTMT2*.

Within Figure 3.5f, a dynamic SQL procedure is shown for the condition when a new table is generated that may be appended or dynamically edited based upon the available information. This condition is often referred to as the *null* database for appending new information.

```
CREATE PROCEDURE drop_table (table_name IN VARCHAR2) AS
  cid INTEGER;
  rv INTEGER;
BEGIN
  /* Open new cursor and return cursor ID. */
  cid := DBMS_SQL.OPEN_CURSOR;
  /* Parse and immediately execute dynamic SQL statement built by
  concatenating table name to DROP TABLE command. */
  DBMS_SQL.PARSE(cid, 'DROP TABLE ' || table_name, dbms_sql.v7);
  /* Close cursor. */
  DBMS_SQL.CLOSE_CURSOR(cid);
EXCEPTION
  /* If an exception is raised, close cursor before exiting. */
  WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cid);
    RAISE;
END drop_table;
```

Figure 3.5f: Creating a Dynamic Drop Table. The drop table is used to create a new (blank table) for inserting materials data. The algorithm produces an output of the following format:

```
SQL> exec drop_table('test tbl')
PL/SQL procedure successfully completed.

SQL> SELECT COUNT(*) from test tbl
ORA-00942:table or view does not exist
```

For this example, the output simply indicates that the procedure has been executed and the table is dropped from the current database. As an alternative methodology, the table may be modified or edited when the number of database columns is not known a priori. Techniques for operating on the database under this scenario are shown in Figure 3.5g. For this case example an internal call to *g_number_of_colums* is used to dynamically extract the number of columns (database variables) for the query.

```

create or replace package my_dbms_sql
as
  procedure define_all( p_cursor in integer );
  type varchar2_table is table of varchar2(4000) index by
    binary_integer;
  function fetch_row( p_cursor in integer )
    return varchar2_table;
end;
/

create or replace package body my_dbms_sql
as
  g_number_of_columns dbms_sql.number_table;
  procedure define_all( p_cursor in integer )
  as
    l_columnValue varchar2(4000);
    l_descTbl dbms_sql.desc_tab;
    l_colCnt number;
  begin
    dbms_sql.describe_columns( p_cursor,
                              l_colCnt, l_descTbl );
    for i in 1 .. l_colCnt loop
      dbms_sql.define_column( p_cursor, i, l_columnValue, 2000 );
    end loop;
    g_number_of_columns(p_cursor) := l_colCnt;
  end;
  function fetch_row( p_cursor in integer )
    return varchar2_table
  is
    l_return varchar2_table;
  begin
    for i in 1 .. g_number_of_columns(p_cursor) loop
      l_return(i) := NULL; dbms_sql.column_value( p_cursor, i, l_return(i) );
    end loop;
    return l_return;
  end;
end;

declare
  l_theCursor integer default dbms_sql.open_cursor;
  l_status integer;
  l_data my_dbms_sql.varchar2_table;

  procedure execute_immediate( p_sql in varchar2 )
  is
  BEGIN
    dbms_sql.parse(l_theCursor,p_sql,dbms_sql.native);
    l_status := dbms_sql.execute(l_theCursor);
  END;
begin
  execute_immediate( 'alter session set nls_date_format=
    ''dd-mon-yyyy hh24:mi:ss''-' );
  dbms_sql.parse( l_theCursor,
    replace( 'select * from test2_tbl',
      '...', '||' ),
    dbms_sql.native );
  my_dbms_sql.define_all( l_theCursor );
  l_status := dbms_sql.execute(l_theCursor);
  while ( dbms_sql.fetch_rows(l_theCursor) > 0 ) loop
    l_data := my_dbms_sql.fetch_row( l_theCursor );
    for i in 1 .. l_data.count loop
      dbms_output.put_line( l_data(i) );
    end loop;
  end loop;

end;
/

```

Figure 3.5g: Creating a Dynamic Table when the Number of Columns is Not Known. For this condition, the database is queried on a case-by-case basis to determine the number of variables that is reflected in the RDBMS.

3.6 Creating Unique Data Records in the LIMS

One technique to ensure uniqueness of a table's primary key is by using sequence numbers. The Oracle/LIMS sequence number provides a method for generating a series of distinct numbers that are tagged to a LIMS measurement of laboratory testing sequence. The sequence is assigned a unique identifier with a starting value, minimum index, maximum index and an incremental value. The assignment technique guarantees unique identification values. As shown in the laboratory demonstrations, the identifiers are used to conform to ISO standards for material identification. In addition, these methods are used to create the chain-of-custody tools that have been developed for LIMS QA/QC. Figure 3.6a demonstrates one method for creating a unique sequence number using a primary key.

```
CREATE SEQUENCE NEW_SEQUENCE
START WITH 1
INCREMENT BY 1
MINVALUE 0

Declare
Machine_id number;
Machine_name varchar2(100):='New Machine';

Stmt varchar2(1000);
BEGIN
stmt:='INSERT INTO test2_tbl
      (machine_id,machine_name,measure_id
      )
      VALUES (new_sequence.nextval, :name,
              :cost
              )';

      execute immediate stmt using machine_name,machine_id;
commit;
end;
/
```

Figure 3.6a: Creating a Unique Instrument Identification Value (Machine ID). In this example, a new sequence is created that begins with an initial value equal to one, and is incremented for each subsequent measure. The procedure *New_Sequence* is used to insert a record into a database table. The sequence created is accessed by the command *<sequence name>.nextval*. Each time this command is executed, a unique value will be generated and inserted into the database. As previously illustrated, the *bind* variables are used to populate the SQL statement. The algorithm produces an output of the following format:

```
SQL> select * from test2_tbl where machine_id=1;

MACHINE_ID      MACHINE_NAME      MEASURE_ID
-----
1              New Machine      25045682

SQL> select * from test2_tbl where machine_id=2;

MACHINE_ID      MACHINE_NAME      MEASURE_ID
-----
2              New Machine      25046784
```

For this example, two measurement identification values are created from two independent queries. The first measurement id = 25045682 corresponds to the first instrument (machine id=1), whereas, the second query generates a measurement id = 25046784 for the next instrument in the sequence. In Figure 3.6b, methods for immediate execution are provided. These techniques combine the earlier definitions for parse, bind, execute and close within a single operation. As a result, they should be used only in discrete operations that do not require extreme duplication. For example, the *EXECUTE IMMEDIATE* statement should not be used within long looping operations since the parsing, binding, and closing operations would create significant delays in the database operation and subsequent query. Within this illustration sections are crossed-out to show the compact nature of the *EXECUTE IMMEDIATE* statement. As indicated, the parsing and binding operations are not required since the single statement replaces a sequence of individual operations.

```

PROCEDURE insert_into_table (table_name varchar2,
                             deptnumber number,
                             deptname varchar2,
                             location varchar2)
IS
  stmt_str varchar2(200);
cur_hdl integer;
rows_processed binary_integer;
BEGIN
  stmt_str := 'insert into ' || table_name ||
              ' values (:deptno, :dname, :loc);';
cur_hdl := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(cur_hdl, stmt_str, dbms_sql.native);
DBMS_SQL.BIND_VARIABLE(cur_hdl, ':deptno', deptnumber);
DBMS_SQL.BIND_VARIABLE(cur_hdl, ':dname', deptname);
DBMS_SQL.BIND_VARIABLE(cur_hdl, ':loc', location);
rows_processed := DBMS_SQL.EXECUTE(cur_hdl);
DBMS_SQL.CLOSE_CURSOR(cur_hdl);

  EXECUTE IMMEDIATE stmt_str USING deptnumber, deptname, location;
END;

```

Figure 3.6b: Using the *EXECUTE IMMEDIATE* Statement to Simplify Database Operations. Statements are shown as crossed-out conditions that are no longer required by the SQL process, since the single *EXECUTE IMMEDIATE* statement compactly processes the information within a single declaration.

Two sample applications for the *EXECUTE IMMEDIATE* statement are provided in Figure 3.6c-d. In these examples, SQL statements are generated at runtime (in real-time during the query process), and implemented to *INSERT* or *UPDATE* a table of machine data. In the first example (Figure 3.6c), the values of three variables, *MACHINENUM*, *MACHINENAME* AND *ROCKWELL_HARDNESS* are inserted into *test2_tbl*. The SQL statement, as shown by the variable *STMT*, is created at run time and executed using the *EXECUTE IMMEDIATE* statement. The benefit of this method is that the user gains all of the advantages of Dynamic SQL within an abbreviated format.

```

DECLARE
MACHINENUM NUMBER:=999;
MACHINENAME VARCHAR2(100):='NEW MACHINE';
ROCKWELL_HARDNESS NUMBER:=5;
STMT VARCHAR2(1000):='INSERT INTO APP_TBL VALUES (:1,:2,:3)';
BEGIN
EXECUTE IMMEDIATE STMT USING MACHINENUM,MACHINENAME,ROCKWELL_HARDNESS;
COMMIT;
END;

```

Figure 3.6c: Execution of *INSERT* statement using *EXECUTE IMMEDIATE* and bind variables. As with *DBMS_SQL*, the *INSERT* statement is built at run time in the *STMT* string variable using values passed in as arguments. THE SQL statement held in *STMT* is then executed via the *EXECUTE IMMEDIATE* statement. The bind variables *:1*, *:2*, *:3* are bound to the arguments which, in this case, are the parameters: *MACHINENUM*, *MACHINENAME* and *ROCKWELL_HARDNESS*.

To verify that the procedure has inserted the information into the table correctly, issue a *SELECT* statement and verify the results as correct. The algorithm produces an output of the following format:

```

SQL> select * from app_tbl where machine_id=999;
999 NEW MACHINE 525000

```

Within Figure 3.6d, a case example is provided for using the *EXECUTE IMMEDIATE* process within an update statement. In this algorithm, *MACHINE_NUMBER* 999 is updated to reflect a revised *ROCKWELL_HARDNESS* index of 6. As in the insert statement shown in Figure 3.6c, the SQL statement is created at run time and the database is updated with the *EXECUTE IMMEDIATE* statement using bind variables.

```

DECLARE
MACHINENUM NUMBER:=999;
MACHINENAME VARCHAR2(100):='NEW MACHINE';
ROCKWELL_HARDNESS:=6;
STMT VARCHAR2(1000):='UPDATE APP_TBL SET MACHINE_ID=:1, MACHINE_NAME =:2, MACHINE_COST=:3
WHERE MACHINE_ID=999';
BEGIN
EXECUTE IMMEDIATE STMT USING MACHINENUM,MACHINENAME,MACHINECOST;
COMMIT;
end;
/

```

Figure 3.6d: Execution of an *UPDATE* statement using the *EXECUTE IMMEDIATE* and the Bind Variables. As previously described for *DBMS_SQL*, the *INSERT* statement is built at run time in the *STMT* string using values passed into the procedure (as arguments). THE SQL statement held in *STMT* is then executed via the *EXECUTE IMMEDIATE* statement. The bind variables *:1*, *:2*, *:3* are attached to the arguments (algorithm parameters) shown as: *MACHINENUM*, *MACHINENAME* and the *ROCKWELL_HARDNESS*.

To verify that the procedure functioned as expected, issue a *SELECT* statement to verify the information in the record for *MACHINE_ID* 999 is correct:

```

SQL> select * from app_tbl where machine_id=999;
999 NEW MACHINE 6

```

3.7 Applying Array Structures within the LIMS

Within this discussion, we examine the use and application of array methods for storing localized variable information. Arrays are used to index processes and maintain ordered or sequenced operations including labels, parameters, and pointers to index algorithms (procedures and functions). The most commonly used structure is the *VARARRAY* and the Nested Table.

The *VARARRAY* is an ordered set of data elements, with each element having an index, and all elements being of the same data type or data structure. The size of a *VARARRAY* refers to the maximum number of elements that may be placed within the structure. Within the Oracle RDBMS for materials information, the *VARARRAY* structures are of variable width, but the maximum size of any particular *VARARRAY* type must be specified a priori (i.e. when the *VARARRAY* type is formally declared).

A Nested Table is an unordered set of elements. The ingredients and structure of the nested table can be queried in SQL and is not defined a priori. A nested table is not created with any particular number of rows. Hence, the size is determined in a dynamic manner during execution.

Both *VARARRAY*s and Nested Tables are one-dimensional, although the elements can be complex object types. *VARARRAY* types are used for one-dimensional arrays, while nested table types are used for single-column tables within an outer table. A variable of any *VARARRAY* type can be referred to as a *VARARRAY*, while a variable of any nested table type can be referred to as a nested table. Process steps for using the *VARARRAY* and Nested Table operations are shown in Figures 3.7a-b. Within the first example, the parameter *num_varray* is declared as a *VARARRAY* whose maximum size cannot exceed five elements. The array is then populated with data consisting of the number sequence 10, 20, 30, and 40. The array is then displayed by referencing the array index numbers 1 to 4. Next, the value stored at index 4 is changed from 40 to 60 and the array is re-displayed showing the modification.

```
DECLARE
  Type num_varray is VARARRAY(5)OF NUMBER;
  v_numvarray num_varray;
BEGIN
  v_numvarray :=num_varray(10,20,30,40);
  --Referencing individual elements
  dbms_output.put_line('The elements in the v_numvarray are: ');
  dbms_output.put_line(to_char(v_numvarray(1))||', '||to_char(v_numvarray(2))||
    ', '||to_char(v_numvarray(3))||', '||
    to_char(v_numvarray(4)));

  --assignment
  v_numvarray(4):=60;
  dbms_output.put_line('The elements in the v_numvarray are: ');
  dbms_output.put_line(to_char(v_numvarray(1))||', '||to_char(v_numvarray(2))||
    ', '||to_char(v_numvarray(3))||', '||
    to_char(v_numvarray(4)));
END;
/
```

Figure 3.7a: The *VARARRAY* Single Dimensional Structure. The array element is created and information is stored and retrieved by the array index value -- similar to other array structures used in modern programming languages. The algorithm produces an output of the following format:

```
The elements in the v_numvarray are:
10, 20, 30, 40
The elements in the v_numvarray are:
10, 20, 30, 60
```

The procedure shown in Figure 3.7a may be modified to utilize the nested table data structure. This modification is illustrated in Figure 3.7b. In this example, the formal data is organized within a column orientation inside the table -- without order. When the data is retrieved, the information is presented in an array.

```

declare
  Type num_table is TABLE OF NUMBER;
  v_numarray num_table;
begin
  v_numarray := num_table(10,20,30,40);
  --Referencing individual elements
  dbms_output.put_line('The elements in the v_numarray are: ');
  dbms_output.put_line(to_char(v_numarray(1))||', '||to_char(v_numarray(2))||
    ', '||to_char(v_numarray(3))||', '||
    to_char(v_numarray(4)));

  --assignment
  v_numarray(4):=60;
  dbms_output.put_line('The elements in the v_numarray are: ');
  dbms_output.put_line(to_char(v_numarray(1))||', '||to_char(v_numarray(2))||
    ', '||to_char(v_numarray(3))||', '||
    to_char(v_numarray(4)));
end;
/

```

Figure 3.7b: The Nested Table Data Structure. The array element is created and ordered according to the execution statements. The algorithm produces an output of the following format:

```

The elements in the v_numarray are:
10, 20, 30, 40
The elements in the v_numarray are:
10, 20, 30, 60

```

3.8 LIMS Error Functions and Exception Management

Under unusual operating conditions, the LIMS and RDBMS will create data codes that indicate potential problems in the management of information. This may occur when the instrument is not properly operated by the end-user. Alternatively, error codes may be generated by simple processing of data, that is not properly registered, or identified within the LIMS system. Within the RDBMS, exception codes may be generated as new information or code elements are added. This frequently occurs when new algorithms are tested or modified for final operation.

Within PL/SQL, all errors are trapped and identified as *EXCEPTIONS*. When the analyst tries to incorporate the erroneous information into the LIMS, the normal processing is stopped, and the control is transferred to the exception handling section of the program. The exception handler mechanism allows a clean separation of the error processing code from the executable statements. This process also forces the systems administrator to examine the potential source of the problem and register the solution.

The LIMS user can create an exception condition by trying to access null or void data sets. This occurs when the path to the database is incorrect or the actual database system contains null or void information (i.e. no valid data or data references). Specific exceptions may be managed using the *implicit* cursor. An example for this technique is provided in Figure 3.8a-b. Within Figure 3.8a, the analyst is attempting to access a database with no information (i.e. the cursor attempts to execute a *SELECT* statement that returns no data).

```

Section 1:
DECLARE
RESULT VARCHAR2(50);
begin
SELECT MACHINE_NAME
INTO RESULT
FROM APP_TBL
WHERE MACHINE_ID=9999;
dbms_output.put_line(result);
end;
/

```

When this procedure is executed, no data is found by the SELECT statement and an error is produced as follows:

```

ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 4

```

Section 2:
By inserting an EXCEPTION section in the procedure, the program becomes aware of the potential problem and handles it appropriately.

```

DECLARE
RESULT VARCHAR2(50);
begin
SELECT MACHINE_NAME
INTO RESULT
FROM APP_TBL
WHERE MACHINE_ID=9999;
dbms_output.put_line(result);
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        dbms_output.put_line('No Data Found');
end;
/

```

Now the procedure terminates normally and displays a message to the user indicating that no data was found.

No Data Found

Figure 3.8a: Exception Handling – Implicit Cursor With *NO_DATA_FOUND*. Within Section 1, the implicit cursor finds no data. This error prematurely terminates the program, and generates an ORA-01403 “no data found error”. Within Section 2, the algorithm is modified to trap for *NO_DATA_FOUND*. As a result the error is caught, and a simple message is displayed to notify the end user. Within Section 2, no system error is generated – only a notification to the analyst.

Since an implicit cursor can only handle a single data element, an error will result when the query returns more than a single value. This condition occurs when the analyst is trying to access numerous fields or data arrays without actual knowledge of the underlying data structure. This condition will also occur if the user is trying to manage too many variables or cases within the database. An example of this condition is provided in Figure 3.8b. For this illustration, the user is attempting to access too many rows of information from an unknown table within the CTD materials database. Without error handling the following situation will occur when using an implicit cursor:


```

Section 1:
DECLARE
RESULT VARCHAR2(50);
begin
SELECT MACHINE_NAME
INTO RESULT
FROM APP_TBL;
dbms_output.put_line(result);
end;
/

```

When this procedure is executed, an error is produced:

```

ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4

```

Section 2:
By implementing error handling, and trapping for the *TOO_MANY_ROWS* error, the program terminates normally.

```

DECLARE
RESULT VARCHAR2(50);
begin
SELECT MACHINE_NAME
INTO RESULT
FROM APP_TBL;
dbms_output.put_line(result);
EXCEPTION
    WHEN TOO_MANY_ROWS
    THEN
        dbms_output.put_line('Query Returned More Than One Row');
end;
/

```

Now the procedure terminates normally and displays a message to the user indicating:

```

Query Returned More Than One Row

```

Figure 3.8b: Exception Handling – Implicit Cursor With *TOO_MANY_ROWS*. Within Section 1, the implicit cursor finds more than one row of data. This error prematurely terminates the program, and generates an ORA-01422 “exact fetch returns more than request number of rows” error. Within Section 2, the algorithm is modified to trap for *TOO_MANY_ROWS*. As a result the error is caught, and a simple message is displayed to notify the end user. Within Section 2, no system error is generated – only a notification to the analyst.

When an exception is raised in a PL/SQL block, normal execution is halted and control is transferred to the exception section or the systems administrator. For the case of a serious error or violation, the process is immediately terminated to preserve the information that is being accessed. In some cases, the ability to continue past exceptions is desired. Figure 3.8c demonstrates a method to handle the exception and then continue with the remainder of the procedure. In the first section, the error is trapped. However, the control never returns from the *EXCEPTION* section, and the statement that displays the message “This occurs after the error” is never reached. Within the second section, the control is allowed to exit the *EXCEPTION* section for additional processing and data analysis.

Section 1:

```

DECLARE
RESULT VARCHAR2(50);
COLNAM VARCHAR2(10):='NAME1';
COLVAL VARCHAR2(10):='VALUE5';
STMT VARCHAR2(100):='INSERT INTO TEST3_TBL VALUES(:1,:2)';
BEGIN
    EXECUTE IMMEDIATE STMT USING COLNAM,COLVAL;
    dbms_output.put_line('This occurs after the error');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX
    THEN
        dbms_output.put_line('Column Name Already Used...Please Select Another');
END;
/

```

Column Name Already Used...Please Select Another

Section 2:

To enable the program to continue after the error is handled, a separate block with a BEGIN section is inserted to hold the exception section.

```

DECLARE
RESULT VARCHAR2(50);
COLNAM VARCHAR2(10):='NAME1';
COLVAL VARCHAR2(10):='VALUE5';
STMT VARCHAR2(100):='INSERT INTO TEST3_TBL VALUES(:1,:2)';
BEGIN
    BEGIN
        EXECUTE IMMEDIATE STMT USING COLNAM,COLVAL;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX
        THEN
            dbms_output.put_line('Column Name Already Used...Please Select Another');
    END;
    BEGIN
        dbms_output.put_line('This occurs after the error');
    END;
END;
/

```

Using this method enables the procedure to handle the error and then continue and display the message "This occurs after the error" as shown below.

Column Name Already Used...Please Select Another
This occurs after the error

Figure 3.8c: Continuing Program Execution After An Exception. Within Section 1, the program is terminated as soon as the error occurs. In this case an attempt is made to insert a duplicate value in a field that is restricted with a unique value constraint. The statement "This occurs after the error" is not reached since the program has already terminated. Within Section 2, the program is not terminated, and statement "This occurs after the error" is displayed to the end user. The second example places the exception section in it's own block, and therefore, enables the program to continue after the error has been handled.

Under certain conditions, the analyst will encounter an exception condition that is specific to an application or resource. These errors are localized to the code that is the source of the error and may not be trapped by SQL or PL/SQL. These error conditions can be managed using programmer-defined exceptions as illustrated in Figure 3.8d.

In this example the variable *big_number_exception* is declared and given an *EXCEPTION* data type.

Now the program logic can determine when the declared exception should be raised. The logical criteria to throw the *big_number_exception* is when *cnumber* > 1,000,000. When this criterion is reached, the exception is raised and control is transferred to the *EXCEPTION* block where the exception is managed. No system level error is generated from this condition.

```

DECLARE
big_number_exception EXCEPTION;
anumber number:=20000;
bnumber number:=60000;
cnumber number;
BEGIN
    BEGIN
        cnumber:=anumber*bnumber;
        IF cnumber>1000000 then
            RAISE big_number_exception;
        end if;
    EXCEPTION
        WHEN big_number_exception THEN
            dbms_output.put_line('A Big Number Exception Has Occurred');
    END;
    BEGIN
        dbms_output.put_line('The answer is '||anumber||'*'||bnumber||' = '||cnumber);
    END;
END;
/

```

Figure 3.8d: Error Events within a Custom Application. Within this algorithm, the exception *big_number_exception* has been declared and defined. When the criteria defined for the exception has been met, the exception is executed and the end user is notified of the event without loss of system resources. The algorithm produces an output of the following format:

```

A Big Number Exception Has Occurred
The answer is 20000*60000 = 1200000000

PL/SQL procedure successfully completed.

SQL>

```

3.9 LIMS Mail Resources

Within this section, two algorithms are provided that illustrate common techniques for e-mail management within a secure internal network. The methods may be adapted for external networks, and will operate on common systems that utilize internal messaging procedures (e.g. the CTD LIMS system), as well as, external code resources (standard internet service). In Figure 3.9a, the process will send email from the PL/SQL code. This approach is used to notify system users of specific events or error conditions. In addition, this method may be used to create a bulletin-board environment for the distribution of common data or help resources. Initially, a table of users and their email addresses will be created. This table will provide a list of individuals who will be notified when a specified event occurs. This includes user identification data, names, and e-mail addresses:

```

CREATE TABLE NOTIFY_USERS
(
    USERID          NUMBER,
    USERNAME        VARCHAR2(25 BYTE),
    USER_EMAIL      VARCHAR2(50 BYTE)
)

```

Next the table will be populated with data of users to be notified. This table can be further broken down into classes of users, which messages to send to a specific group of users and other combinations.

```
INSERT INTO NOTIFY_USERS VALUES(1,'USER1','user1@LIMStest.net');
INSERT INTO NOTIFY_USERS VALUES(1,'USER2','user2@LIMStest.net');
Commit;
```

Now the email procedure will be created. This procedure creates a cursor of individuals to notify from the *NOTIFY_USERS* table. Variables are declared with default values for the sender, recipient, subject, mail host and the message to be sent via email. As the cursor loops through user data, it creates an *SMTP* connection to the specified mail server and sends a predefined message to recipients that have been previously identified.

```
CREATE OR REPLACE PROCEDURE email_notification
IS
    sender          VARCHAR2(100) default 'sender@test.net';
    recipient        VARCHAR2(100) default 'recipient@test.net';
    subject          VARCHAR2(100) default 'An event has occurred';
    message          VARCHAR2(100) default 'A record has been inserted into the test3_tbl';
    emailaddress      varchar2(100) default 'address@test.net';
    crlf VARCHAR2(2):= UTL_TCP.CRLF;
    connection        utl_smtp.connection;
    mailhost          VARCHAR2(255) := 'mail.test.net';
    header            VARCHAR2(1000);

    CURSOR users IS
        SELECT username, user_email FROM notify_users;

BEGIN
    for the_result in users LOOP

        connection := utl_smtp.open_connection(mailhost,25);
        header:= 'Date: '||TO_CHAR(SYSDATE,'dd Mon yy')||crlf||
            'From: '||sender||'|'|crlf||
            'Subject: '||subject||crlf||
            'To: '||the_result.user_email;

        -- Handshake with the SMTP server
        utl_smtp.helo(connection, mailhost);
        utl_smtp.mail(connection, sender);
        utl_smtp.rcpt(connection, the_result.user_email);
        utl_smtp.open_data(connection);
        -- Write the header
        utl_smtp.write_data(connection, header);
        utl_smtp.write_data(connection, crlf||crlf||message);
        utl_smtp.close_data(connection);
        utl_smtp.quit(connection);
    end LOOP;
EXCEPTION
    WHEN UTL_SMTP.INVALID_OPERATION THEN
        http.p(' Invalid Operation in SMTP transaction.');
```

```
    WHEN UTL_SMTP.TRANSCIENT_ERROR THEN
        http.p(' Temporary problems with sending email - try again
later.');
```

```
    WHEN UTL_SMTP.PERMANENT_ERROR THEN
        http.p(' Errors in code for SMTP transaction.');
```

```
END email_notification;
/
```

Next create a table that will call the email notification procedure.

```
CREATE TABLE test3_tbl
(
    testid NUMBER,
    testdescription VARCHAR2(100)
)
```

Now create a trigger on the table. This trigger will fire and call the *email_notification* procedure every time a user *INSERTS*, *DELETES* or *UPDATES* information in the *test3_tbl*.

```
CREATE OR REPLACE TRIGGER EVENT_TRIGGER
BEFORE DELETE OR INSERT OR UPDATE
ON TEST3_TBL
FOR EACH ROW
DECLARE

BEGIN
    email_notification;
END ;

/
```

Figure 3.9a: E-Mail Notification within the LIMS/CTD Network. Creating the table of users and their respective e-mail addresses that are local to the laboratory system. The list of users will be sent an email notifying them that an event has occurred.

The procedures in Figure 3.9a are used with the field validation methods shown in Figure 3.9b to ensure that only valid e-mail addresses are utilized. Within Figure 3.9b, logical statements are used to test for the standard construction of an email address. This construction is of the form: (@ symbol, 1 ".", no spaces). If the format conditions are met, the function returns 'TRUE' - indicating that it is a valid email address. If one or more of the criteria is not met, a 'FALSE' or invalid address message is returned.

```
Create or Replace FUNCTION is_EmailValid(pEmail_Text VARCHAR2 DEFAULT NULL)
RETURN Varchar2 is
    returnval Varchar2(10) := 'TRUE';
    atLocation NUMBER;
BEGIN
    -- Check to see if this e-mail address contains the @ symbol
    IF INSTR(pEmail_text, '@', 1) = 0 THEN
        returnval := 'FALSE';
    END IF;
    -- Check to see if this e-mail address contains at least 1 . symbol
    IF INSTR(pEmail_text, '.', 1) = 0 THEN
        returnval := 'FALSE';
    END IF;
    -- Check to insure that there are no spaces in the email address
    IF INSTR(ltrim(rtrim(pEmail_text)), ' ', 1) > 0 THEN
        returnval := 'FALSE';
    END IF;

    IF INSTR(LOWER(pEmail_Text), 'mil', 1) = 0 AND INSTR(LOWER(pEmail_Text), 'com', 1) = 0
    AND INSTR(LOWER(pEmail_Text), 'net', 1) = 0 AND
        INSTR(LOWER(pEmail_Text), 'org', 1) = 0 AND INSTR(LOWER(pEmail_Text), 'edu', 1) = 0
    THEN
        returnval := 'FALSE';
    END IF;

    IF pEmail_Text IS NULL THEN
        returnval := 'FALSE';
    END IF;

    RETURN returnval;
END is_EmailValid;
```

Figure 3.9b: Validate Email Address. This procedure accepts an email address string *pEmail* as an input parameter. If the string matches the logical criteria for valid e-mail addressing, the function returns a 'TRUE' value. Alternatively, a 'FALSE' or invalid result is returned from the logical test.

Executing the procedure produces the following results:

```
SQL> select is_EmailValid('abc@aol.com') from dual;
IS_EMAILVALID( 'ABC@AOL.COM' )
-----
TRUE
```

Note that this utility function only checks for valid structure, but does not search or ping the user account to be certain that it is valid for use. Using Figure 3.9a-b, the systems administrator is able to notify all users within the CTD and LIMS network and attempt to notify any and all users that have a valid e-mail address.

3.10 CTD Web Applications and Network Distribution

Within this discussion, we examine the main case examples for creating *html* (Hyper Text Markup Language) based web applications using either SQL or PL/SQL. The applications are required for the distribution of information using the common html approach and may be employed within an intra-net configuration for the secure distribution of digital information.

The ability to dynamically generate Web pages from database records gives the user an advantage not found in conventional static pages: the capability of viewing real-time data. The example provided in Figure 3.10a, illustrates one method for machine data using html. In this case study, the cursor *machine_info* is declared to retrieve the *machine_id*, *machine_name* and *Rockwell_hardness* index from *test2_tbl*.

HTML is created and embedded into the stored procedure using the *htp* (Hyper Text Procedure) command. This technique sends the html that is generated back to the Web browser to be displayed. After beginning the html with *htp.p* the procedure is written in simple html code.

To retrieve data from the cursor, a *FOR LOOP* is constructed. As seen in the example, every record retrieved is then displayed in html. The '||' concatenation symbol is used to separate html from the PL/SQL variables.

After the *FOR LOOP* is closed and the dynamic data generated and displayed, the page is completed following simple html coding practices.

The stored procedure is accessed following the configuration that was implemented when the web server was installed. It generally follows the form:

```
http://<server name>/pls/<data access descriptor (DAD)>/package.procedurename
```

Within the CTD/LIMS, the server is the Compaq ML-530 disk-array that access the *pls* data description. Once this path is identified, the package, procedure or function is executed from within the web-based environment. Note that this same approach is used internally within the design of the StarLIMS enterprise server. This method, allows MOD to distribute all LIMS data across the secure intra-net provided access privileges are granted at the root privilege level. An identical method may be employed for external networks, such as those employed by MOD for mobile operations. These tools will allow MOD to forward user specified information from field locations using common Internet methods that utilize secure web pages.

```

CREATE OR REPLACE PROCEDURE display_tbl2 IS
//declare and create the cursor
CURSOR machine_info IS
    SELECT machine_id,machine_name,rockwell_hardness
    FROM test2_tbl
    ORDER BY machine_id;
BEGIN
//start the HTML
    http.p('<html>
    <head>
    <title>Test Procedure To Display Table Data</title>
    </head>
    <body bgcolor=#C4D8E2>
    <center>
    <table border=1>
    <td align=center><b> Machine ID</b></td><td align=center><b><b>Machine Name</b></td>
    <td align=center><b>Rockwell Hardness Index</b></td></tr> ');
//begin the FOR LOOP and display the records retrieved from the SQL query
FOR results in machine_info LOOP
    http.p('<tr><td align=center>'||results.machine_id||'</td><td>'||results.machine_name||'</td>
    <td align=center>'||results.rockwell_hardness||'</td></tr> ');
end loop;
//end the HTML page
    http.p(' </table></center>
    </body></html>
    ');
END display_tbl2;
/

```

Figure 3.10a: Creating Dynamic *html* Web Pages from within PL/SQL. In this example, the *html* code is generated from information that is a priori stored within the CTD Materials Database. The technique sends the *html* that is generated back to the Web browser to be displayed. The algorithm generates an output of the following format:

Machine ID	Machine Name	Rockwell Hardness Index
0	New Machine	5
1	Machine 1	6
2	Machine 2	7
3	Machine 3	4

As indicated, the result is a simple table of material values indexed by machine identifier. The table presentation is common *html* format for the distribution across the CTD/LIMS environment. The procedure may be modified to create an across-the-web notification utility. For example, a new record may be added into the LIMS or CTD materials database, and all users can be instantaneously notified of the event. The notification sequence is critical to certain testing operations when erroneous measures are corrected or new information is added into the test sequence that will significantly influence the analytical process. This event occurs when outlier information is added into the data record that significantly affects the statistical QA/QC procedures employed for the sample. Within Figure 3.10b-c, a process sequence is illustrated for inserting new records into a web environment and notifying all CTD/LIMS users of the event. To simplify the case example, the package *test_package* contains two procedures: (1) *display_tbl2*, which is used to display the results of the database query, and (2) *update_database*, which is used to perform the database operations on the data in *tbl2*.

```

CREATE OR REPLACE PACKAGE test_package AS
PROCEDURE display_tbl2;
PROCEDURE update_database(
    the_type in varchar2 default 'INSERT',
    machineid in integer default null,
    machinename in varchar2 default null,
    rockwell in integer default null);

END test_package;

/

CREATE OR REPLACE PACKAGE BODY test_package AS
PROCEDURE display_tbl2 IS
    nextnum number;
    CURSOR machine_info IS
        SELECT machine_id,machine_name,rockwell_hardness
        FROM test2_tbl;
BEGIN
    SELECT MAX(MACHINE_ID)+1 INTO
        NEXTNUM
        FROM TEST2_TBL;

    http.p('<html>
<head>
<title>Test Procedure To Display Table Data</title>
</head>
<body bgcolor=#C4D8E2>
<form action=test_package.update_database>
<center>
<table border=1>
<td align=center><b> Machine ID</b></td><td align=center><b><b>Machine Name</b></td>
<td align=center><b>Rockwell Hardness Index</b></td></tr> ');
    for results in machine_info LOOP
    http.p('<tr><td align=center>'||results.machine_id||'</td><td>'||results.machine_name||'</td>
<td align=center>'||results.rockwell_hardness||'</td></tr> ');
    end loop;

    http.p('
<td align=center>'||nextnum||'<input type=hidden name=machineid value='||nextnum||'></td>
<td align=center><input type=text name=machineid value='||nextnum||'></td><td align=center>
<input type=text name=rockwell></td></tr>
<td colspan=3 align=center><input type=submit value="Update Database"></td></table></center>
</form></body></html>
');
END display_tbl2;

```

Figure 3.10b: Inserting Records into the CTD/LIMS Database. The database is updated with information entered into the *html* text boxes. While the procedure is executing, the user is shown a message screen with system status. After the operation is completed, the original page is displayed showing the updated information.

The procedure created in the first example is modified first to ensure every record has a unique identifier (i.e. the next sequentially numbered *machine_id*). This is accomplished by selecting the maximum machine identification code currently in the database, and incrementing it by 1. That result is stored in the variable *nextnum*.

After the current information is displayed, blank input boxes are generated to hold information that will be inserted by the user within the database. When the *html* form is submitted, the values contained in the input boxes are sent to the location specified in the form action. In this example, they are sent to the *update_database* procedure for processing.

When update database receives the values it inserts them into the database. This procedure also utilizes a Java script to perform a redirect, and send the page from the *update_database* procedure to the *display_tbl2* procedure displaying the newly entered data.

```

PROCEDURE update_database(
    the_type in varchar2 default 'INSERT',
    machineid in integer default null,
    machineid in integer default null,
    machineid in integer default null,
    machineid in integer default null)
is
stmt varchar2(100);
BEGIN
    http.p('<html><head><title></title><script>
        function doRedirect()
        {
            document.write('Database Updated...Please Wait');
            setTimeout("window.location =
''test_package.display_tbl2'', 2000);
        }
    </script></head>');
    --
    http.p('<body onload="doRedirect();">');
    IF the_type='INSERT' THEN
        stmt:='INSERT INTO TEST2_TBL values(:1,:2,:3)';

        EXECUTE IMMEDIATE STMT USING MACHINEID,MACHINENAME,ROCKWELL;

        COMMIT;
    END IF;
    http.p('</body></html>');
end update_database;

END test_package;

/

```

Figure 3.10b (Continued): Inserting Records into the CTD/LIMS Database. This procedure operates with the code shown in Figure 3.10a data input and data display using the common html interface. This section provides the update utilities that are required to broadcast the information across the CTD/LIMS network. After the operation is completed, the original page is displayed showing the updated information.

Example operations are shown in Figure 3.10c-e as they appear to the web client. In the first example, the original screen is shown with the inclusion of a blank insertion box for machine identification code four.

Machine ID	Machine Name	Rockwell Hardness Index
0	New Machine	5
1	Machine 1	6
2	Machine 2	7
3	Machine 3	4
4	<input type="text"/>	<input type="text"/>
<input type="button" value="Update Database"/>		

Figure 3.10c: Simple Data Insertion for the CTD/LIMS web client.

Machine ID	Machine Name	Rockwell Hardness Index
0	New Machine	5
1	Machine 1	6
2	Machine 2	7
3	Machine 3	4
4	Machine 4	5

Update Database

Figure 3.10c: User Defined Parameters. Manually updating the database.

Machine ID	Machine Name	Rockwell Hardness Index
0	New Machine	5
1	Machine 1	6
2	Machine 2	7
3	Machine 3	4
4	Machine 4	5
5		

Update Database

Figure 3.10d: Including the New Insertion Point. The user is notified that the database has been updated to include the fourth measure (from Machine 4) shown in Figure 3.10c. Next, the web GUI is automatically updated to show all valid measurements and a new insertion point for entering additional data from Machine ID 5.

The procedures shown in Figure 3.10a-b may be extended to support data reduction and editorial review of the inserted information. This provides the foundation methods for compacting the database to reduce the storage volume and shows how the systems administrator can support interactive data editing features. This method is also used when the MOD analyst attempts to add information that is invalid. The line insertion is initially shown to the user and the case is then removed for further analysis within the QA/QC procedures.

The main structure for automatically editing the database table is shown in Figure 3.10e-f. On the main *html* form, a hyperlink and icon are added to the structure adjacent to each record. When the user clicks on the link, the control passes to the *update_database* procedure with the parameters *the_type=DELETE* and the *machineid* of the specific record that was selected.

When the *update_database* procedure receives this information it determines that it is a delete request and proceeds to delete the record algorithm. After the operation is completed control is passed back to *display_tbl2* and the updated records displayed.

```

CREATE OR REPLACE PACKAGE test_package AS
PROCEDURE display_tbl2;
PROCEDURE update_database(
    the_type in varchar2 default 'INSERT',
    machineid in integer default null,
    machine_name in varchar2 default null,
    rockwell in integer default null);

END test_package;
/

CREATE OR REPLACE PACKAGE BODY test_package AS
PROCEDURE display_tbl2 IS
    nextnum number;
CURSOR machine_info IS
    SELECT machine_id,machine_name,rockwell_hardness
    FROM test2_tbl;

BEGIN

    SELECT MAX(MACHINE_ID)+1 INTO
    NEXTNUM
    FROM TEST2_TBL;

    http.p('<html>
    <head>
    <title>Test Procedure To Display Table Data</title>
    </head>

    <body bgcolor=#C4D8E2>
    <form action=test_package.update_database>
    <center>
    <table border=1>

    <td align=center><b> Machine ID</b></td><td align=center><b><b>Machine Name</b></b></td>
    <td align=center><b>Rockwell Hardness Index</b></td><td
    align=cener><b>Delete</b></td></tr> ');

    for results in machine_info LOOP
    http.p('<tr><td
    align=center>'||results.machine_id||'</td><td>'||results.machine_name||'</td>
    <td align=center>'||results.rockwell_hardness||'</td><td>

    <a
    href="test_package.update_database?the_type=DELETE&machineid='||results.machine_id||'"><im
    g border=0 src="http://webtools.symbolx.com/temporary/trash2.png" ></a></td></tr> ');
    end loop;

    http.p('

    <td align=center>'||nextnum||'<input type=hidden name=machineid value='||nextnum||'></td>
    <td align=center><input type=text name=machine_name></td><td align=center>
    <input type=text name=rockwell></td></tr>

    <td colspan=4 align=center><input type=submit value="Update
    Database"></td></table></center>
    </form></body></html>
    ');

END display_tbl2;

```

Figure 3.10e: Editing and Deleting Records within the CTD/LIMS Database. When the user clicks on the trash icon next to a record the *update_database* procedure deletes the specified record, gives the user a status report that the database is being updated, then redirects the GUI to the original page shown with updated information. After the operation is completed, the original page is displayed showing the updated database records.

```

PROCEDURE update_database(
  the_type in varchar2 default 'INSERT',
  machineid in integer default null,
  machineName in varchar2 default null,
  rockwell in integer default null)
is
  stmt varchar2(100);
BEGIN
  http.p('<html><head><title></title><script>
    function doRedirect()
    { document.write('Database Updated...Please Wait');
      setTimeout("window.location =
''test_package.display_tbl2''", 2000);
    }
  </script></head>');

  http.p('<body onload="doRedirect();">');
  IF the_type='INSERT' THEN
    stmt:='INSERT INTO TEST2_TBL values(:1,:2,:3)';
    EXECUTE IMMEDIATE STMT USING MACHINEID,MACHINENAME,ROCKWELL;
    COMMIT;
  ELSIF the_type='DELETE' THEN
    stmt:='DELETE FROM TEST2_TBL WHERE MACHINE_ID=:1';
    EXECUTE IMMEDIATE STMT USING MACHINEID;
  END IF;
  http.p('</body></html>');
end update_database;

END test_package;
/

```

Figure 3.10f: Editing and Deleting Records within the CTD/LIMS Database (Continued). This procedure operates with the code shown in Figure 3.10e data input and data display using the common *html* interface. This section provides the update utilities that are required to broadcast the editorial revision across the CTD/LIMS network. After the operation is completed, the original page is displayed showing the updated information.

The delete operation is shown in Figure 3.10g. In this example, Machine ID = 0 has been deleted from the main CTD materials database and a blank insertion box for machine identification appears in the *html* interface.





Machine ID	Machine Name	Rockwell Hardness Index	Delete
1	Machine 1	6	
2	Machine 2	7	
3	Machine 3	4	
4	Machine 4	5	
5	<input type="text"/>	<input type="text"/>	
<input type="button" value="Update Database"/>			

Figure 3.10g: Manually Editing Database Records Using Web-Client Interface

The functions and procedures shown in Figure 3.10a-g can be organized to build a single utility that provides the technical support for: data editing, error detection, and user feedback (to the analyst or the systems administrator). This process is organized as a comprehensive utility for email notification with a hyperlink to specific users, clients, and analytical instruments. The algorithm processes information in a sequential manner:

The first step is to create a table of the individuals that will be notified when a report is submitted. The table entitled: *BUG_DISTRIBUTION_TBL* contains columns for the username and email address. Additional columns may be added with user classifications and data parameters for certification and attestation. At the minimum level, the table includes information concerning the user ID, the user name, and the e-mail address. Additional restriction of privileges may be added to the data record during this step:

```
CREATE TABLE BUG_DISTRIBUTION_TBL
(
  NOTIFIED_ID      INTEGER                NOT NULL,
  USERNAME         VARCHAR2(100 BYTE),
  EMAIL_ADDRESS    VARCHAR2(500 BYTE)
)
```

Next the table is populated with sample data from the CTD instruments. Alternatively, archived data may be used from the materials database or the Sybase LIMS RDBMS:

```
INSERT INTO BUG_DISTRIBUTION_TBL VALUES(1,'TESTPERSON','TESTPERSON@TEST.COM');
```

A table is created to hold information about the feedback or bug event. The columns include:

BUG_ID	a number to identify the event
BUG_DATE	date the issue was created
BUG_NAME	user supplied name for the issue
EMAIL	user supplied email address so the user can be contacted
SHORTDESCRIPTION	a user supplied short description of the issue
FULLDESCRIPTION	a detailed description of the problem
DUPLICATION	user provided example of how the problem can be duplicated
WORKAROUND	user provided example of how to work around the problem
IDEALLY	user provided statement explaining ideally how the problem should be resolved
BROWSER	user provided information explaining what Web browser they are using
OPERATINGSYSTEM	user reported information concerning the operating system used
SEVERITY	user assigned ranking of how important the issue is
PRIORITY	user assigned ranking of the priority of the issue
RESOLVED	has the issue been resolved
RESOLVEDBY	who resolved the issue
RESOLVEDDATE	date the issue was resolved

Next, a series of declarations and arrays are required to manage the information. The arrays are required to parse and locally store the information as it is being processed. The tables are also required to hold information about the feedback event. The columns include:

```

CREATE TABLE BUG_REPORT_TBL
(
  BUG_ID          INTEGER,
  BUG_DATE        DATE,
  BUG_NAME        VARCHAR2(100 BYTE),
  EMAIL           VARCHAR2(100 BYTE),
  SHORDESCRIPTION VARCHAR2(100 BYTE),
  FULLDESCRIPTION VARCHAR2(4000 BYTE),
  DUPLICATION     VARCHAR2(4000 BYTE),
  WORKAROUND      VARCHAR2(4000 BYTE),
  IDEALLY         VARCHAR2(4000 BYTE),
  BROWSER         VARCHAR2(100 BYTE),
  OPERATING_SYSTEM VARCHAR2(100 BYTE),
  SEVERITY        VARCHAR2(100 BYTE),
  PRIORITY        VARCHAR2(100 BYTE),
  RESOLVED        VARCHAR2(10 BYTE),
  RESOLVEDBY      VARCHAR2(50 BYTE),
  RESOLVEDDATE    DATE
)

```

Next a sequence is created to assign a unique ID to each bug report.

```

CREATE SEQUENCE BUG_SEQUENCE
START WITH 0
INCREMENT BY 1
MINVALUE 0
NOCACHE
NOCYCLE
NOORDER

```

All prior work is committed to the database.

```
COMMIT;
```

Then a package is created to contain the procedures that will be used in the example. For simplicity, all the procedures are included within a single package. Within the CTD-LIMS environment, this example would be broken into multiple packages that could easily be shared by numerous applications that are internal (or external) to the CTD laboratory complex.

Next, the package header is declared. This lists all of the procedures and functions that will be used within the newly created package:

```
CREATE OR REPLACE PACKAGE suggestions AS
```

Once the package is created, the *LoadCSS* utility is used to define the Cascading Style Sheets (CSS) that are utilized by other CTD procedures. Using this method enables the user to change the appearance of the *html* tags from a single location. Many different styles are presented in this procedure although they are not all used in the example:

```
PROCEDURE LoadCSS;
```

After loading the CSS, an *STMP* connection is established. This procedure initiates the *STMP* with a user defined mail server, and generates an email message to users that are listed within the *BUG_DISTRIBUTION_TBL* table. A hyperlink is then sent to each user giving them a path to display the referenced issue. The procedure accepts parameters that include: the sender of the message, the recipient, the subject and the email message body:

```

PROCEDURE MAIL
(
sender      IN VARCHAR2,
recipient   IN VARCHAR2,
subject     IN VARCHAR2,
message     IN VARCHAR2
);

```

Following the *STMP* connection, a procedure is developed to update the table: *BUG_REPORT_TBL* to show that the issue is resolved. This includes time and date stamping for when the issue was corrected and indicating the analyst that performed the service. This process mirrors the formal chain-of-custody procedures used within the main LIMS shell:

```

PROCEDURE RESOLVED(bug_id in integer default null,
                  username in varchar2 default 'DEVELOPER1');

```

Following the issue resolved procedure, a display utility is used to notify the user of the problem and provide the ancillary information including the system identification code for the error:

```

PROCEDURE display(id in integer default null);

```

The main procedure is the starting point of the application. It displays the option of viewing the list of issues, or entering a new issue.

```

Procedure main;

```

This procedure updates or modifies the database for information entered or modified on the input form. The algorithm also provides the user with the capability to edit information as it is entered into the system. The parameters are the columns of *BUG_REPORT_TBL*:

```

PROCEDURE modifybug(
bug_name      in varchar2 default null,
email         in varchar2 default null,
shortdescription  varchar2 default null,
fulldescription   varchar2 default null,
duplication     varchar2 default null,
workaround      varchar2 default null,
ideally         varchar2 default null,
browser         varchar2 default null,
operating_system varchar2 default null,
severity        varchar2 default null,
priority        varchar2 default null,
bugid          in number);

```

Following the parameter declarations, the errors may be viewed and recorded using the *viewbug* procedure:

```

PROCEDURE viewbug(id in integer);

```

This procedure displays the information recorded for all issues in the form of a report. For example:

```

PROCEDURE ShowIssues;
END suggestions;
/

```

After the package specification, the procedures and functions are created.

```

CREATE OR REPLACE PACKAGE BODY suggestions AS CTD Main Global Variables
*****

```

The global variables should be modified to reflect information specific to the web environment and should be the only modifications necessary to run the full application. The variables include the *mailhost* as the address of the mail server (used for notification of all technical data). In this example, *emailfrom* is the person or organization sending the notifications:

```
mailhost      varchar2(50):='mail.test.net';
server        varchar2(50):='http://www.test.com/pls/<Database Access Descriptor>;
emailfrom     varchar2(50):='testperson@abc.net';
```

As previously noted, the *LoadCSS* procedure then creates the Cascading Style Sheets used within the *html* tags of the procedures. The process is shown in Figure 3.11a :

```
PROCEDURE LoadCSS
IS
BEGIN
  http.p('<style type="text/css">');
  http.p('
body {font-family:"Arial"; font-size:"9 pt"; color:Black; }
H1 {font-family:"Arial"; font-size:"20 pt"; color:Navy; }
H2 {font-size:"8 pt"; }
H3 {font-family:"Arial"; font-size:"10 pt"; color:Navy; }
H4 {font-family:"Arial"; font-size:"10 pt"; color:Red; }

P {}
TR {}
TD {color:black; font-size:"8 pt"; }

TH {font-family:"Arial"; font-size:"8 pt"; color:Navy; }
.fld {font-size:"8 pt"; color:Black; background-color:"#a0a0a0"; border:"White Thin";
padding:"2pt"; }
.val {font-size:"8 pt"; color:Black; background-color:"#AACAFF"; margin-left:"0 pt"; margin-
right:"0 pt"; border:"White Thin"; padding:"1 pt"; }
.tbsum {font-size:"8 pt"; color:White; background-color:"#064406"; border:"White Thin";
margin-left:"0 pt"; margin-right:"0 pt"; border:"White Thin"; padding:"1 pt"; }
border:"White Thin"; padding:"1 pt"; margin-left:"0 pt"; margin-right:"0 pt";}
.tab {font-size:"8 pt"; text-align:"Center"; color:White; background-color:"#0000cc";
padding:"2 pt"; }
.reporttab {font-size:"7 pt"; text-align:"Center"; color:White; background-color:"#064406";
.plainboldtext {font-size:"8 pt"; font-weight:bold; color:Black; background-color:White; }
.plainboldtext_big {font-size:"10 pt"; font-weight:bold; color:Navy; background-color:White; }
.plainwhite {font-size:"8 pt"; color:White; background-color:White; }
.menu {font-size:"8 pt"; text-align:"center"; alink:"#00ff00"; background-color:"#ffffff";
padding:"2 pt";}
.report {font-size:"8 pt"; color:Black; background-color:"#0000ff"; text-align:"Left"; }
.navybg {font-size:"8 pt"; color:White; background-color:"#000066"; text-align:"Center"; }
.sansa {
  font-family: Arial, Helvetica, sans-serif
}

#tree lnk{
  color: blue;
  text-decoration: underline;
}

');
  http.p('</style>');
End LoadCSS;
```

Figure 3.11a: Creating the Cascading Style Sheets. This procedure illustrates many of the *html* text related processes including the setup graphics. The example has been condensed for illustration purposes -- since the actual sheet may require many pages of scripting.

The dynamics for the mail forwarding procedure are shown in Figure 3.11b. In this example, the main *SMTP* processes are shown in a condensed format. The procedure is required to contact the main CTD database server to forward information concerning new data structures, potential errors, and

ancillary or metadata records. The mail procedure is select contained and requires no input arguments. The algorithm format is shown below:

```

PROCEDURE mail
(
  sender      IN VARCHAR2,
  recipient   IN VARCHAR2,
  subject     IN VARCHAR2,
  message     IN VARCHAR2

) IS
  emailaddress varchar2(100);
  stmt          varchar2(100);
  type theRefCursor is Ref Cursor;
  gCursor theRefCursor;

  crlf VARCHAR2(2) := UTL_TCP.CRLF;
  connection utl_smtp.connection;
  mailhost varchar2(50) := suggestions.mailhost;
  header VARCHAR2(1000);

BEGIN
  stmt := 'select email_address from bug_distribution_tbl';
  -- Start the connection.
  open gCursor for stmt;

Loop
  Fetch gCursor into emailaddress;
  exit when gCursor%NOTFOUND;
  connection := utl_smtp.open_connection(mailhost,25);
  header := 'Date: ' || TO_CHAR(SYSDATE, 'dd Mon yy hh24:mi:ss') || crlf ||
    'From: ' || sender || crlf ||
    'Subject: ' || subject || crlf ||
    'To: ' || emailaddress;
  -- Handshake with the SMTP server
  utl_smtp.helo(connection, mailhost);
  utl_smtp.mail(connection, sender);
  utl_smtp.rcpt(connection, emailaddress);
  utl_smtp.open_data(connection);
  -- Write the header
  utl_smtp.write_data(connection, header);
  utl_smtp.write_data(connection, crlf || crlf || message);
  utl_smtp.close_data(connection);
  utl_smtp.quit(connection);
end loop;
close gCursor;

EXCEPTION
  WHEN UTL_SMTP.INVALID_OPERATION THEN
    http.p(' Invalid Operation in SMTP transaction. ');
  WHEN UTL_SMTP.TRANSPORT_ERROR THEN
    http.p(' Temporary problems with sending email - try again
later. ');
  WHEN UTL_SMTP.PERMANENT_ERROR THEN
    http.p(' Errors in code for SMTP transaction. ');
END;

procedure RESOLVED(bug_id in integer default null,
                  username in varchar2 default 'DEVELOPER1')
IS
  ID INTEGER DEFAULT NULL;
BEGIN
  ID := BUG_ID;
  update bug_report_tbl
  set resolved='Yes',
      RESOLVEDBY=USERNAME,
      RESOLVEDDATE=SYSDATE
  WHERE BUG_ID=ID;
COMMIT;

```

Figure 3.11b: Accessing the *SMTP* Server. This procedure illustrates the access points for the main notification of the edit report within the CTD network.

The creation of style sheets and the mail notification is controlled from within the LIMS graphical user interface. The form interface and the look and feel of the LIMS are controlled by the form update sequence. A sample form update is shown in Figure 3.11c-e. Within Figure 3.11c, the main body of the update sequence is provided for the *SMTP* server access and the network message utilities. This includes the distribution of data and metadata records for the support of digital instruments in the fixed and mobile laboratories:

```

procedure updateform(
    bug_name in varchar2 default null,
    email in varchar2 default null,
    shortdescription varchar2 default null,
    fulldescription varchar2 default null,
    duplication varchar2 default null,
    workaround varchar2 default null,
    ideally varchar2 default null,
    browser varchar2 default null,
    operating_system varchar2 default null,
    severity varchar2 default null,
    priority varchar2 default null,
    bugid number default null
)
is
    stmt varchar2(4000);
    type gcursor_type is ref cursor;
    theseq integer;
    gcursor gcursor_type;
begin
    STMT:=' insert into bug_report_tbl
(bug_id,bug_date,bug_name,email,shortdescription,fulldescription,duplication,workaround,ideall
y,browser,operating_system,
    severity,priority)
values(bug_sequence.nextval,:1,:2,:3,:4,:5,:6,:7,:8,:9,:10,:11,:12) returning bug_id into
:13';
    execute immediate stmt using
    sysdate,bug_name,email,shortdescription,fulldescription,duplication,workaround,ideally,
    browser,operating_system,severity,priority returning into theseq;
    commit;

    suggestions.mail(''||emailfrom||','
    '||emailfrom||','Bug
    Report/Suggestion',''||suggestions.server||'/suggestions.viewbug?id='||theseq||');

    http.p('<script language=JavaScript>
    setTimeout("window.location = ''suggestions.main'', 2000);
    document.write('<b>Your report has been submitted</b>');
    </script>');
    end;

    PROCEDURE MAIN
    IS
    BEGIN
        http.p('<html><head><title>Bug/Suggestion Reporting</title></head>
        <body><center><h1>Bug/Suggestion Reporting</h1>
        <table border=1>
        <td align=center><input type=button value= "Create A Bug/Suggestion Report"
        onClick=window.location="''||suggestions.server||'/SUGGESTIONS.display"></td></tr>
        <td align=center><input type=button value= "View Bug/Suggestion Reports"
        onClick=window.location="''||suggestions.server||'/SUGGESTIONS.ShowIssues"></td></tr>
        </table></center>
        </body>
        </html>');
    END MAIN;

```

Figure 3.11c: The Form Update Sequence. This procedure illustrates the main process steps for updating a mobile or fixed laboratory client using forms for data input and data archive. The procedure also manages the error notification sequence within the LIMS network.

```

procedure modifybug(
  bug_name in varchar2 default null,
  email in varchar2 default null,
  shortdescription varchar2 default null,
  fulldescription varchar2 default null,
  duplication varchar2 default null,
  workaround varchar2 default null,

  ideally varchar2 default null,
  browser varchar2 default null,
  operating_system varchar2 default null,
  severity varchar2 default null,
  priority varchar2 default null,
  bugid      in number

)
is
  stmt varchar2(4000);
  type gcursor_type is ref cursor;
  theseq      integer;
  gcursor     gcursor_type;

begin
  STMT:=' update bug_report_tbl set bug_date=:1, bug_name=:2, email=:3, shortdescription=:4,
  fulldescription=:5, duplication=:6,

  workaround=:7, ideally=:8, browser=:9, operating_system=:10, severity=:11 ,priority=:12
  where bug_id=:13';

  execute immediate stmt using
  sysdate,bug_name,email,shortdescription,fulldescription,duplication,workaround,ideally,
  browser,operating_system,severity,priority,bugid ;
  commit;

  suggestions.mail(suggestions.emailfrom,
  suggestions.emailfrom,'Bug
  Reporting/Suggestion','||suggestions.server||'/suggestions.viewbug?id='||bugid||');

  http.p('<script>

  setTimeout("window.location = ''suggestions.main'', 2000);
  document.write('<b>Your report has been updated</b>');
  </script>');
  end;

```

Figure 3.11d: The Form Update Sequence – Error Notification. The *modifybug* procedure is used to organize the error notification message into a format that is easily understood by the client users within the LIMS network. The procedure also performs the system updates that are required to notify the LIMS administrators. The information is sent to a pool of specified users and may contain specific metadata records to document the solution procedures. This procedure is a continuation of the form update sequence shown in Figure 3.11c.

The notification procedures continue using a compact notation to display unique error codes or metadata identification values. The codes are unique integer values that are used to identify and document the source of the error process as it occurs within the CTD materials database or the main LIMS network. The display process and the management methods for the unique identification codes are shown in Figure 3.11e-f. Within Figure 3.11e, the primary declarations are shown including the indexing methods for determining the severity of the error process. The procedure also shows the related *html* that is required to format and display all operations:

```

PROCEDURE display(id in integer default null)
IS
  BUG_DATE# DATE default sysdate;
  BUG_NAME# VARCHAR2(100) default null;
  EMAIL# VARCHAR2(100) default null;
  SHORTDESCRIPTION# VARCHAR2(100) default null;
  FULLDESCRIPTION# VARCHAR2(4000) default 'When I ...';
  DUPLICATION# VARCHAR2(4000) default 'To replicate the problem ...';
  WORKAROUND# VARCHAR2(4000) default 'To get rid of the problem ...';
  IDEALLY# VARCHAR2(4000) default 'Ideally this should ...';
  BROWSER# VARCHAR2(100) default null;
  OPERATING_SYSTEM# VARCHAR2(100) default null;
  SEVERITY# VARCHAR2(100) default null;
  PRIORITY# VARCHAR2(100) default null;
BEGIN
  suggestions.loadcss;
  if id is not null then
    SELECT
      BUG_DATE#,BUG_NAME#,EMAIL#,SHORTDESCRIPTION#,FULLDESCRIPTION#,DUPLICATION#,WORKAROUND#,IDEALLY#,
      BROWSER#,OPERATING_SYSTEM#,SEVERITY#,PRIORITY#
    INTO
      BUG_DATE#,
      BUG_NAME#,
      EMAIL#,
      SHORTDESCRIPTION#,
      FULLDESCRIPTION#,
      DUPLICATION#,
      WORKAROUND#,
      IDEALLY#,
      BROWSER#,
      OPERATING_SYSTEM#,
      SEVERITY#,
      PRIORITY#
    FROM BUG_REPORT_TBL
    WHERE BUG_ID=ID;
  end if;
  suggestions.loadcss;
  http.p('<HEAD><TITLE>Suggestion Reporting Tool</TITLE></HEAD>');
  http.p('<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF0000" ALINK="#FF0000"
  VLINK="#FF0000" topmargin="0" leftmargin="10" marginheight="0" rightmargin="0"
  marginwidth="0">');

  if id is null then
    http.p('<FORM METHOD="POST" ACTION="suggestions.updateform">');
  else
    http.p('<FORM METHOD="POST" ACTION="suggestions.modifybug">');
  end if;

  http.p('<table width=60% border=0 cols=2 align=center>
  <td align="center" style="font-size: 12pt;">Bug Reporting/Suggestion Form</td><td
  align=center>
  <input type=button value="Return To Main Page"
  onClick=window.location="||suggestions.server||/suggestions.main"></td>
  </tr><tr>
  <td class=hfld colspan=2 style="font-size: 11pt;">Personal Details</td></tr>
  <tr>
  <td class=val width=100><B>Name</B></td><td class=val> <INPUT TYPE="text" NAME="bug_name"
  value="||bug_name#||" size=50 maxlength=80></td></tr>
  <tr><td class=val> <B>E-mail address</B></td><td class=val><INPUT TYPE="text" NAME="email"
  value="||email#||" size=50 MAXLENGTH=80></td></tr>
  <tr><td class=val> <B>Date</B></td><td class=val>||to_char(bug_date#,'MONTH-DD-
  YYYY')||</td></tr>
  <tr><td class=hfld colspan=2 style="font-size: 11pt;">Problem</td></tr>
  <tr><td class=val><B>Short description</B></td><td class=val><INPUT TYPE="text"
  NAME="shortdescription" value="||SHORTDESCRIPTION#||" size=80 MAXLENGTH=80> </td></tr>
  <tr><td class=val><P><B>Full description</B><BR>
  <TEXTAREA NAME="fulldescription" ROWS=5 COLS=50>||FULLDESCRIPTION#||</TEXTAREA></td>
  <td class=val><P><B>Describe how to replicate the problem</B><BR>
  <TEXTAREA NAME="duplication" ROWS=5 COLS=50>||DUPLICATION#||</TEXTAREA></td></tr>

```

Figure 3.11e: The Form Display Process – Section 1. This algorithm shows the primary declarations and the organization of the required *html* code to display the metadata records and resultant error codes from CTD materials database and the main LIMS server. This procedure continues with the code resources shown in Figure 3.11f.

```

<tr><td class=val><P><B>Describe any workarounds you have found</B><BR>
<TEXTAREA NAME="workaround" ROWS=5 COLS=50>'||WORKAROUND#||'</TEXTAREA></td>

<td class=val><P><B>Describe how you would like the application to function</B><BR>
<TEXTAREA NAME="ideally" ROWS=5 COLS=50>'||IDEALLY#||'</TEXTAREA></td></tr>

<tr><td class=hfld style="font-size: 11pt;">Severity</td>
<td class=hfld style="font-size: 11pt;">Priority</td></tr>');

if SEVERITY# = 'Usability' then
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Usability" checked>');
else
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Usability"> ');
end if;
http.p('Usability problem<BR></td>');

if PRIORITY#='ASAP' then
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="ASAP" checked>');
else
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="ASAP">');
end if;
http.p('Must be fixed as soon as possible</td></tr>');

if severity# = 'Annoyance' then
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Annoyance" checked>');
else
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Annoyance">');
end if;
http.p('Annoyance</td>');

if PRIORITY#='Must Fix' THEN
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="Must Fix" CHECKED>');
ELSE
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="Must Fix" >');
END IF;
http.p('Must fix</td></tr>');

if severity# = 'Cosmetic' then
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Cosmetic" checked>');
else
http.p('<tr><td><INPUT TYPE="radio" NAME="severity" VALUE="Cosmetic">');
end if;
http.p('Cosmetic problem</td>');

IF PRIORITY#='When Possible' then
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="When Possible" checked>');
else
http.p('<td><INPUT TYPE="radio" NAME="priority" VALUE="When Possible" >');
end if;
http.p('Fix if time allows</td></tr>

<tr><td class=hfld colspan=2 style="font-size: 11pt;">Software and Hardware</td></tr>
<tr><td class=val>What kind of browser are you using?
<select NAME="browser">
<option>Internet Explorer 6.0</option>
<option>Netscape 7.0</option>
<option>Other</option>
</select></td>
<td class=val>What kind of operating system are you using?
<select NAME="operating_system">
<option>Windows</option>
<option>Linux</option>
<option>Other</option>
</select>
</td></tr>

```

Figure 3.11f: The Form Display Process – Section 2. This algorithm shows the organization of the *html* code to display the metadata records and resultant error codes from CTD materials database and the main LIMS server. The code uses the standard Explorer browser to show the results to the MOD analyst. This process continues with the code resources shown in Figure 3.11g.

```

<tr><td><P>');
http.p('<input type=hidden name="bugid" value="'||id||'>');

if id is null then
  http.p('<INPUT TYPE="submit" VALUE="Submit">');
else
  http.p('<INPUT TYPE="submit" VALUE="Update Report">');
end if;

  http.p('<P></td></tr></FORM><HR></table></BODY>  ');
END;

procedure viewbug(id in integer)
is
  BUG_DATE# DATE;
  BUG_NAME# VARCHAR2(100);
  EMAIL# VARCHAR2(100);
  SHORTDESCRIPTION# VARCHAR2(100);
  FULLDESCRIPTION# VARCHAR2(4000);
  DUPLICATION# VARCHAR2(4000);
  WORKAROUND# VARCHAR2(4000);
  IDEALLY# VARCHAR2(4000);
  BROWSER# VARCHAR2(100);
  OPERATING_SYSTEM# VARCHAR2(100);
  SEVERITY# VARCHAR2(100);
  PRIORITY# VARCHAR2(100);

BEGIN
  http.print('<html><script>
    function goHome(){
      window.open(''||suggestions.server||'.suggestions.main', TARGET="_self" );
    }
    function problemRes(){

window.open(''||suggestions.server||'/suggestions.resolved?bug_id=''||id||'&username=DEVELOPER1
);", TARGET="_self" );
    }
  </script>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF0000" ALINK="#FF0000" VLINK="#FF0000"
topmargin="0" leftmargin="0" marginheight="0" rightmargin="0"
marginwidth="0"
>
  ');
  http.p('<table width=100% cellpadding=0 cellspacing=0 border=0><tr><td align="center"
cellpadding=0>
</td></tr>
<tr><td> ');
    suggestions.loadcss;
SELECT
BUG_DATE,BUG_NAME,EMAIL,SHORTDESCRIPTION,FULLDESCRIPTION,DUPLICATION,WORKAROUND,IDEALLY,
  BROWSER,OPERATING_SYSTEM,SEVERITY,PRIORITY
INTO
  BUG_DATE#,
  BUG_NAME#,
  EMAIL#,
  SHORTDESCRIPTION#,
  FULLDESCRIPTION#,
  DUPLICATION#,
  WORKAROUND#,
  IDEALLY#,
  BROWSER#,
  OPERATING_SYSTEM#,
  SEVERITY#,
  PRIORITY#
FROM BUG_REPORT_TBL
WHERE BUG_ID=ID;

```

Figure 3.11g: The Form Display Process – Section 3. This algorithm continues the main process steps shown in Figure 3.11e-f, using *html* code to display the metadata records, and resultant error codes from CTD materials database and the main LIMS server.

```

http.p('<html><body>
  <table width=50% align=center><tr><td class=val>Date</td><td
class=val>'||to_char(bug_date#,'MONTH-DD-YYYY')||'</td></tr>
<tr><td class=val>Name</td><td class=val>'||bug_name#||'</td></tr>
  <tr><td class=val>Email</td><td class=val><a
href="mailto:'||email#||'">'||email#||'</a></td></tr>

  <tr><td class=val>Short Description</td><td class=val>'||shortdescription#||'</td></tr>
  <tr><td class=val>Full Description</td><td class=val>'||fulldescription#||'</td></tr>
  <tr><td class=val>Duplication</td><td class=val>'||duplication#||'</td></tr>
  <tr><td class=val>Work Around</td><td class=val>'||workaround#||'</td></tr>
  <tr><td class=val>Ideally</td><td class=val>'||ideally#||'</td></tr>
  <tr><td class=val>Browser</td><td class=val>'||browser#||'</td></tr>

  <tr><td class=val>Operating System</td><td class=val>'||operating_system#||'</td></tr>
  <tr><td class=val>Severity</td><td class=val>'||severity#||'</td></tr>
  <tr><td class=val>Priority</td><td class=val>'||priority#||'</td></tr>
  <tr><td class=val align=center>');

  http.p('<input type=button value="Problem Resolved" onClick="problemRes();">');

http.p('</td><td class=val align=center><input type=button value="Go Back"
onClick="javascript:history.back(1);">');"></td>
</tr>

</body></html>');
end;

Procedure ShowIssues
IS
bugid                number;
bugname              varchar2(100);
bugdate              date;

bugdesc              varchar2(100);
bugseverity           varchar2(100);
bugpriority           varchar2(100);
bugresolved           varchar2(50);
bugresolvedby         VARCHAR2(100) default 'Developer1';
bugdateresolved       DATE;

stmt                 varchar2(2000);
type iCursorType is  ref cursor;
iCursor              iCursortype;
myclass               varchar2(10);

BEGIN

stmt:='SELECT b.bug_id, b.bug_name, b.bug_date, b.shortdescription, b.severity, b.priority,
b.resolved, b.resolvedby, b.resolveddate
FROM bug_report_tbl b';

open iCursor for stmt;

suggestions.loadcss;
http.p('<html><head></head><title>Suggestion Reporting Tool</title><br>');
  http.p('<table border=1 borderColor=#999999><tr><td width=500 class=hfld align=center><a
style="font-family: Tahoma; font-size: 9pt; color: navy;"
href="'||package_init.schema||'.suggestions.display">Click to Create a New Suggestion / Bug
Report</a></td></tr></table>');

```

Figure 3.11h: The Form Display Process – Section 4. This algorithm continues the main process steps shown in Figure 3.11e-g, using *html* code to display the metadata records, and resultant error codes from CTD materials database and the main LIMS server. The main display features are terminated and the procedures for showing the technical reports and related metadata records are initiated. The *ShowIssues* procedure manages the suggestion process for editing and appending new information.

```

http.p('<TABLE id="reportTable" BORDER: black 1px solid; WIDTH: 99%; font-size : 7pt;
background-color:#bbd6bb;"
borderColor=#999999 cellSpacing=0 cellPadding=2 border=1 dragcolor='gray' slcolor=#88ff88
hlcolor=#eeeecc > ');
http.p('<thead><TR align=center >
<TD width=140 class=hfld style="font-family: Tahoma; font-size: 7pt;" nowrap
align=center>Name</TD>
<TD width=80 class=hfld style="font-family: Tahoma; font-size: 7pt;" nowrap
align=center>Date</TD>
<TD width=250 class=hfld style="font-family: Tahoma; font-size: 7pt;" align=center>Short
Description</TD>
<TD width=80 class=hfld style="font-family: Tahoma; font-size: 7pt;" nowrap
align=center>Priority</td>
<td width=80 class=hfld style="font-family: Tahoma; font-size: 7pt;" nowrap
align=center>Resolved</td>
<td width=20 class=hfld style="font-family: Tahoma; font-size: 7pt;" nowrap
align=center>Modify</td></tr>
');
Loop
Fetch iCursor into bugid, bugname, bugdate, bugdesc, bugseverity, bugpriority, bugresolved,
bugresolvedby, bugdateresolved;
if iCursor%ROWCOUNT<=0 THEN
    http.print ('<tr><td class="litenp"><p align="justify"><FONT FACE="Times New
Roman" SIZE="4" COLOR="#3300CC">No Data Available</font></p></td>
<td class=litenp></td>
<td class=litenp></td>
<td class=litenp></td>
<td class=litenp></td>
<td class=litenp></td>
<td class=litenp></td>
</tr>');
END IF;
exit when iCursor%NOTFOUND;
if mod(iCursor%ROWCOUNT,2)=0 then myclass:='lite';

else
myclass:='dark';
end if;

http.p('<tr>');
http.p('<td align=right abbr='||bugname||' class='||myclass||'><a target=_parent
href="||package_init.schema||'.suggestions.viewbug?id=||bugid||'
onMouseOver="status='Click To View Bug Report : '";
return true"

onMouseOut="status=' '";return true"
title="Click To View Bug Report: ">||bugname||</a></td>');
http.p('<td align=right abbr='||to_char(bugdate,'J')||' class='||myclass||'><a target=_parent
href="||package_init.schema||'.suggestions.viewbug?id=||bugid||'
onMouseOver="status='Click To View Bug Report : '";return true"
onMouseOut="status=' '";return true"
title="Click To View Bug Report: ">||to_char(bugdate,'DD-MON-YYYY')||</a></td>');
if bugdesc is null then
http.p('<td align=right abbr=z class='||myclass||'>--</td> ');
else
http.p('<td align=right abbr='||bugdesc||' class='||myclass||'><a target=_parent
href="||package_init.schema||'.suggestions.viewbug?id=||bugid||'
onMouseOver="status='Click To View Bug Report : '";return true"
onMouseOut="status=' '";return true"
title="Click To View Bug Report: ">||bugdesc||</a></td>');
end if;

```

Figure 3.11i: The Form Display Process – Section 5. This algorithm continues the main process steps shown in Figure 3.11e-h, using *html* code to display the metadata records, and resultant error codes from CTD materials database and the main LIMS server. This section adds *onMouseOver* events to update the window status when the user hovers over the reference position *href*. Tool tips are also included in the title documentation.


```

if bugseverity is null then
http.p('<td align=right  abbr=z class='||myclass||'>--</td> ');
else
http.p('<td align=right abbr='||bugseverity||' class='||myclass||'><a target=_parent
href='||package_init.schema||'.suggestions.viewbug?id='||bugid||'
onMouseOver="status='Click To View Bug Report :  '
return true"
onMouseOut="status='  '";return true"
title="Click To View Bug Report: ">'||bugseverity||'</a></td>');
end if;
if bugpriority is null then
http.p('<td align=right  abbr=z class='||myclass||'>--</td> ');
else
http.p('<td align=right abbr='||bugpriority||' class='||myclass||'><a target=_parent
href='||package_init.schema||'.suggestions.viewbug?id='||bugid||'
onMouseOver="status='Click To View Bug Report :  '
return true"
onMouseOut="status='  '";return true"
title="Click To View Bug Report: ">'||bugpriority||'</a></td>');
end if;
if bugresolved is null then
http.p('<td align=right abbr="no" class='||myclass||'><a target=_parent
href='||package_init.schema||'.suggestions.viewbug?id='||bugid||'
onMouseOver="status='Click To View Bug Report :  '
return true"
onMouseOut="status='  '";
return true"
title="Click To View Bug Report: ">No</a></td>');
else
http.p('<td align=right abbr='||bugresolved||' class='||myclass||'><a target=_parent
href='||package_init.schema||'.suggestions.viewbug?id='||bugid||'
onMouseOver="status='Click To View Bug Report :  '
return true"
onMouseOut="status='  '";
return true"
title="Click To View Bug Report: ">'||bugresolved||'</a></td>');
end if;
if bugresolvedby is null then
http.p('<td align=right  abbr=z class='||myclass||'>--</td> ');
else
http.p('<td align=right abbr='||bugresolvedby||' class='||myclass||'><a target=_parent
href='||package_init.schema||'.suggestions.viewbug?id='||bugid||'
onMouseOver="status='Click To View Bug Report :  '";return true"
onMouseOut="status='  '";return true"
title="Click To View Bug Report: ">'||bugresolvedby||'</a></td>');
MON-YYYY')||'</a></td>');
end if;
http.p('<td align=right abbr="z" class='||myclass||'><a target=_parent
src='||package_init.Schema||'.utility.deliver_media?graphics_name=flashlight_trans.gif"
border="0" vspace="5"></a></td>');
end loop;
http.p(' </tbody></table></html>');
end;
END suggestions;
/

```

Figure 3.11j: The Form Display Process – Section 6. This algorithm continues the main process steps shown in Figure 3.11e-i, using *html* code to display the metadata records, and resultant error codes from CTD materials database and the main LIMS server. This section completes the web-based application and graphical user interface.

The procedures and algorithms shown in Figure 3.11a-j create a dynamic environment for the management of digital information within the CTD fixed and mobile laboratory system. The applications are used to generate custom reports and manage specific events or error conditions. For the management of error codes and *bug*-conditions, the LIMS uses a dialog of the form shown in Figure 3.12a. This form is also used to enter information and notify LIMS users of error conditions based upon 12th Main Directorate priority standards.

Bug Reporting/Suggestion Form [Return To Main Page](#)

Personal Details

Name:

E-mail address:

Date: MARCH -05-2004

Problem

Short description:

Full description: Describe how to replicate the problem
When I ... To replicate the problem ...

Describe any workarounds you have found: To get rid of the problem ... Describe how you would like the application to function: Ideally this should ...

Severity

☐ Usability problem

☐ Annoyance

☐ Cosmetic problem

Priority

☐ Must be fixed as soon as possible

☐ Must fix

☐ Fix if time allows

Software and Hardware

What kind of browser are you using? Internet Explorer 6.0

What kind of operating system are you using? Windows

[Submit](#)

Figure 3.12a: Managing Error Codes and *Bug*-Conditions within the CTD Fixed and Mobile Laboratory Environment. The dialog is shown as it appears on the web server and may be used by the mobile teams on common laptops under Microsoft Explorer.

[Click to Create a New Suggestion / Bug Report](#)

Name	Date	Short Description	Severity	Priority	Resolved	Resolved By	Date Resolved	Modify
A Test User	05-MAR-2004	A description of My Problem	Annoyance	When Possible	No	--	--	✎

Figure 3.12b: Managing Error Reports and Metadata Records. The simple interface is used to access technical records and error events as they are published within the CTD network. Records cannot be modified at the user level without permission from the senior LIMS administrator.

The metadata records are displayed to the analyst on a case-by-case basis using simple hyperlinks to show how the issue has been resolved and to assist MOD in the identification of the ancillary

information. This includes the standard chain-of-custody information as shown in Figure 3.12b. The sample hyperlinks for this process are shown in Figure 3.12c. In this example, an error sequence is identified and the user is notified via e-mail hyperlinks. Within the MOD laboratory, intranet links are used to maintain security standards, however, the procedures are generalized to include standard e-mail addressing with support for common messaging services (such as AOL).

Date	MARCH -05-2004
Name	A Test User
Email	testuser@aol.com
Short Description	A description of My Problem
Full Description	When I ...
Duplication	To replicate the problem ...
Work Around	To get rid of the problem ...
Ideally	Ideally this should ...
Browser	Internet Explorer 6.0
Operating System	Windows
Severity	Annoyance
Priority	When Possible
<div>Problem Resolved</div> <div>Go Back</div>	

Figure 3.12c: Error Records and Hyperlinks to Ancillary Data. The sheet description (hyperlink shown in Figure 3.12b) is used to access additional metadata records shown in this illustration. All information is nested by hyperlink access and tied to specific LIMS users with local e-mail addresses. This process is required to conform to MOD standards for chain-of-custody within the intranet environment.

As the issues are resolved, the metadata records are updated and the chain-of-custody records are modified to indicate how the solution has been implemented within the CTD network. As shown in Figure 3.12d, the resolved record is displayed for developer-1 with the ancillary time-date indicators for the resolution event.

Click to Create a New Suggestion / Bug Report								
Name	Date	Short Description	Severity	Priority	Resolved	Resolved By	Date Resolved	Modify
A Test User	05-MAR-2004	A description of My Problem	Annoyance	When Possible	Yes	DEVELOPER1	05-MAR-2004	

Figure 3.12d: Resolved Error Records. As issues are resolved and approved by MOD, the solution is displayed within the CTD network and indexed by the analyst (that resolved the issue) and the technical description of the problem (registered by time and date).

3.11 CTD Instrumentation and Maintenance Documentation

The CTD laboratory system is composed of fixed and mobile laboratory instruments that are networked within the LIMS using SQL and PL/SQL methods for the distribution of technical data, primary measurements, and metadata records. The instruments are organized in a tabular format shown in Figure 3.13a-b by reference item, instrument manufacturer, and measurement function.

Within the LIMS system, this table is used to organize all tests that have been performed within a known date using hyperlinks to the manufacturer and the respective function. Using this technique, detailed service and maintenance records can be documented for all instruments in the CTD complex.

Item	Instrument	Measurement Function
F1.1	Philips PW2404 XRF Spectrometer	1
F2.1	Shimadzu AG-250kNG	2
F2.2	Shimadzu Servo Pulser EHF-EG	3
F2.3	Charpy 300J Impact Tester	4
F3.1	Leica DM-IR (MEF4M) Inverted Microscope	5
F3.2	Vickors (Sony) MHT-10 Imaging System	5
F3.3	Leica Q550MW Digital IP System	5
F3.4	Leica MZ6 stereomicroscopes	5
F3.5	Struers LaboPress/RotoPol21/RotoForce3	6
F3.6	Struers Consumables	6
F4.1	Ernst AT130 DR-T Hardness Tester	4
F5.1	Philips XL30 W-TMP Electron Microscope	1,5

Figure 3.13a: LIMS *Fixed* Laboratory Equipment. 1=primary element detection and spectral analysis, 2=tensile and compression testing, 3=cyclic testing for tension, compression, and material ductility, 4=primary hardness detection, 5=optical measurement of metallographic specimen, 6=sample preparation.

As shown in Figure 3.13a, the CTD Fixed laboratory system is composed of complex instruments for spectrographic examination of samples using scanning electron methods as well as x-ray fluorescence techniques. The instrumentation also utilizes standard optics for surface evaluation and enhanced optical evaluation methods for image processing and materials classification. The laboratory is composed of non-destructive testing methods (measurement functions 1, 5) as well as destructive methods for investigating tension, compression, and surface-sub-surface ductility (measurement functions 2,3).

The instrumentation used in the CTD Mobile laboratory system is documented in Figure 3.13b. In this table, the methods for examining field in situ samples are provided by vendor specific instrument and destructive (non-destructive) application. As shown, the main systems utilize a non-destructive technology (ultrasonic amplification, optical magnification, chemical or capillary amplification, and magnetic particle amplification) to determine and evaluate the test specimen. In addition, a flash-arc methodology is shown in item 1.1 for primary element identification. This technique is surface destructive, but does not damage the main sub-surface structure. The mobile laboratory also contains instruments for hardness determination using both destructive and non-destructive methods for sample evaluation at the surface level. The laboratory includes two systems for network evaluation of complex piping and longitudinal lifting systems (item 2.1.1 and item 5.1). Each system employs a series of networked transducers to determine the modal characteristics of the field system. For the Intraspect system, this includes specialized software for scanning linear sections such as long pipes used in high-pressure boilers. For the A-Line 32D system, specialized software is provided for triangularization. This method assists the user in determining the source of the flaw within a complex network such as long pipes that are joined in multiple sections. The remaining systems are shown by

their respective measurement functions for laboratory support. The Intros system is shown as item A.1 since this unit was added into the laboratory to address specific requirements for cable deformation and elongation (loss of cross-section) that is required to examine crane-lifting cables. This process was required for Gosgorteknadzor certification. The final two systems (Intek VES4-10-300 and Stresstel FlawMikes) are formally installed on the Pomoshnik search-and-rescue vehicles. These systems are commonly used by MOD mobile field teams for on-site diagnostics. They perform similar functions to the SiteScan 230 Flaw detectors (item 2.1) and the Intek Endoscope Systems (item 7.1).

Item	Instrument	Measurement Function
1.1	ARC-MET930SP (Spectrum 18 OE Spectrometer)	1
2.1	SiteScan 230 Flaw Detector	2
2.1.1	Intraspect NT Imaging Systems	3
3.1	StressTel T-Mike EL	4,5
3.7	Sonatest Alphagage A-Scan	4,5
4.1	ESX/B 310PD Mag Inspector (Parker B310BDC-A)	6
4.2	ESX/PM 50 (Parker PM50-A)	6
5.1	A-Line 32D 12 Channel Acoustic Emission System	3
6.1	HardTip 2000 (Proceq Equotip Model D)	7
7.1	Intek Endoscope Systems	8
8.1	Visual Inspection Systems (Vert, Russia)	8
9.1	Residual Stress Tester	9
10.1	EFI 300 Leak Detector	10
11.1	Testo 625 Hygrometer	11
12.1	NP-600 Hydraulic Press	12
13.1	Start 1M Compressor	12
14.1	HBM Dynamometer	13
15.1	Wika Pressure Transmitter	10
16.1	Mega-Check Thickness Gages	5
17.1	Capillary Flaw Equipment and Materials	14
18.1	Concrete Flaw Detector	2
19.1	Engineering Level w/ Tripod	12
20.1	Insulation Resistance Meter	15
21.1	Ground Resistance Meter	15
A.1	Intros Magnetic Cable Tester	16
P.1	Pomoshnik: Intek VES4-10-300 Video Endoscopes	8
P.2	Pomoshnik: StressTel FlawMikes & Transducers	4,5

Figure 3.13b: LIMS *Mobile* Laboratory Equipment. 1=primary element detection and spectral analysis, 2=NDT-UT (non-destructive testing, ultrasonic) flaw detection, 3=network analysis using NDT-UT, 4=NDT-UT surface hardness, 5=NDT-UT material thickness, 6=magnetic particle flaw detection, 7=surface hardness (pin destructive testing), 8=optical measurement metallographic examination, 9=NDT residual stress analysis of sub-surface structure, 10=pressure detection–vacuum detection, 11=barometric pressure–hydrographic measurements, 12=sample preparation and laboratory support, 13= dynamic torque and acceleration, 14=capillary and chemical amplification of surface features, 15=physical resistance or conductance characterization, 16=magnetic field detection and sub-element characterization.

Within the CTD LIMS environment, archive methods have been developed for organizing the service and maintenance records for all fixed and mobile laboratory instruments. The interface for this search utility is shown in Figure 3.14a. In this example, the analyst begins a query for all tests that have been performed using the SITESCAN instrumentation. Note that the user may query the database using partial strings, and does not need to search on SITESCAN 230 Flaw Detector or the exact naming convention.

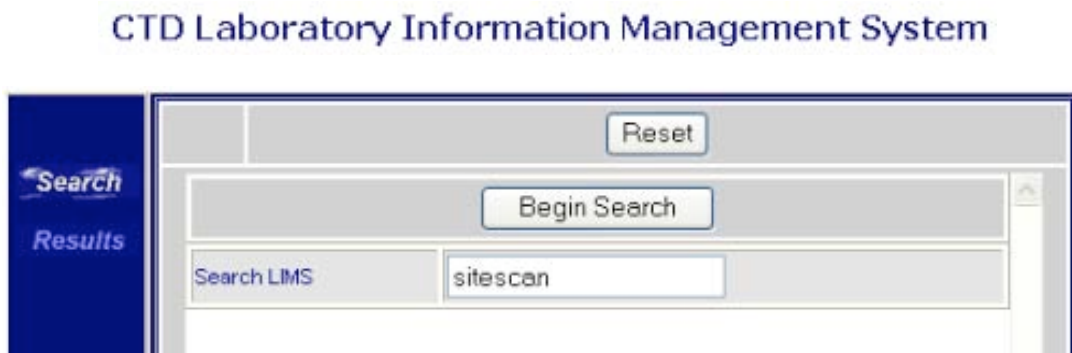


Figure 3.14a: LIMS Maintenance Records – Search Utility. This illustration shows the basic layout for the search utility that operates within the CTD LIMS and the CTD Materials Database. The search tool was written using SQL and PL/SQL. Hence this archive and maintenance utility will operate on any SQL compliant database found within the MOD laboratory system. As shown, the user enters the search string “sitescan” and presses “Begin Search” button to examine the MOD database for records that match the search criteria.

The application searches the database *lab_equipment* and returns, as hyperlinks, information that matches the search criteria. When the user selects one of the returned hyperlinks, they are taken to a separate page that lists, in greater detail, information pertaining to the item selected.

The application utilizes dynamic *html*, JavaScript, and PL/SQL to retrieve database information and display it as a Web page. The algorithms that support the search utility are shown in Figure 3.14b-f. Each algorithm is provided in a small section with documentation that is required to describe the process. In the final working utility, all algorithms are combined to create the working model. In Figure 3.14b, the data declarations are provided for the table that displays the laboratory equipment (maintenance records or test results). This table is organized by time, instrument, measurement technique, or other function:

```
CREATE TABLE LAB_EQUIPMENT
(
  ITEM          VARCHAR2(50 BYTE),
  INSTRUMENT    VARCHAR2(500 BYTE),
  MEASUREMENT_FCN VARCHAR2(10 BYTE),
  FM           VARCHAR2(10 BYTE)
)
```

Figure 3.14b: LIMS Maintenance Records – Table Creation. This illustration shows the size information for the table layout. The table is created as a repository for the information from the LIMS or CTD Materials Database.

The sample data is placed in the table from within the LIMS process during the testing sequence. The data is also added by MOD on a periodic basis to document the maintenance and service records within the CTD. Within Figure 3.14c, a manual insertion is shown for populating the data records. This is provided to illustrate one method for data insertion. Within the CTD materials database, this operation is automated by equipment type -- based upon the data logging capability of the instrument.

```

INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F1.1', 'Philips PW2404 XRF Spectrometer', '1', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F2.1', 'Shimadzu AG-250kNG', '2', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F2.2', 'Shimadzu Servo Pulser EHF-EG', '3', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F2.3', 'Charpy 300J Impact Tester', '4', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F3.1', 'Leica DM-IR (MEF4M) Inverted Microscope', '5', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'F3.2', 'Vickors (Sony) MHT-10 Imaging System', '5', 'F');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (

:::::

'13.1', 'Start 1M Compressor', '12', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'14.1', 'HBM Dynamometer', '13', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'15.1', 'Wika Pressure Transmitter', '10', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'16.1', 'Mega-Check Thickness Gages', '5', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'17.1', 'Capillary Flow Equipment and Materials', '14', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'18.1', 'Concrete Flaw Detector', '2', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'19.1', 'Engineering Level w/ Tripod', '12', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'20.1', 'Insulation Resistance Meter', '15', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'21.1', 'Ground Resistance Meter', '15', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'A.1', 'Intros Magnetic Cable Tester', '16', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'P.1', 'Pomoshnik: Intek VES4-10-300 Video Endoscopes', '8', 'M');
INSERT INTO SYMBOLX.LAB_EQUIPMENT ( ITEM, INSTRUMENT, MEASUREMENT_FCN,
FM ) VALUES (
'P.2', 'Pomoshnik: StressTel FlawMikes & Transducers', '4,5', 'M');
commit;

```

Figure 3.14c: LIMS Maintenance Records – Manual Data Insertion. This illustration shows one method for populating the database with equipment records of the form shown in Figure 3.13a,b. In the actual LIMS process, the data is automatically inserted following the formal QA/QC process.

The query process begins with the *search_start* utility shown in Figure 3.14d. In this example, the search page and the result page are simultaneously loaded into memory. When the user begins a query, the results are instantaneous (moving back and forth between the panels). The two panels (*panel2* and *panel3*) are initially populated with procedures *SEARCH.DISPLAY* and *QUERYBLANK.DISPLAY*. JavaScript procedures are utilized to show or hide the panels, to highlight the “Search” and “Results” buttons when the user rolls over the image and to update the window status when the user moves the mouse over either button.

```
CREATE OR REPLACE PACKAGE search_start AS

PROCEDURE display;
    PROCEDURE donothing;
END search_start;
/

CREATE OR REPLACE PACKAGE BODY search_start AS

PROCEDURE display
IS
    crlf VARCHAR2( 2 ):= CHR( 13 ) || CHR( 10 );
BEGIN
    http.p('<head>
<script language="JavaScript">
    var currentPanel;

function showPanel(panelNum) {
    //hide visible panel, show selected panel,
    //set tab
    if (currentPanel != null) {
        hidePanel();
    }
    document.getElementById
        ('panel'+panelNum).style.visibility = 'visible';
    document.getElementById
        ('stab'+panelNum).style.visibility = 'visible';
    document.getElementById
        ('htab'+panelNum).style.visibility = 'hidden';
    currentPanel = panelNum;
    //setState(panelNum);
    window.name="SEARCHWINDOW";
}

function mouseOn(imagex){
    document.getElementById(imagex).src=imagex+'_OVER.GIF';
}

function mouseOff(imagex){
    document.getElementById(imagex).src='imagex+'.GIF';
}

function hidePanel() {
    //hide visible panel, unhighlight tab
    document.getElementById
        ('panel'+currentPanel).style.visibility =
        'hidden';
    document.getElementById
        ('stab'+currentPanel).style.visibility = 'hidden';
    document.getElementById
        ('htab'+currentPanel).style.visibility = 'visible';
}

```

Figure 3.14d: LIMS Maintenance Records – Searching the Database. This illustration shows the initial section of the query-search utility that provides maintenance support functions for the MOD. The algorithm uses the function *showPanel* to set the visibility of the display. This is accomplished by the command *document.getElementById(elementname).style.visibility='visible' or 'hidden'*. The *mouseOn* function is used to exchange the image with a new distinct view, when the user passes a mouse over the table. It is accomplished by changing the source of the image that is passed to the function as a parameter setting. The *mouseOff* function is the opposite of the *mouseOn* function which returns the image to the original icon when the mouse is moved away. The *hidePanel* function hides the panel display as required.

The buttons and controls are created in the `<div>` tags shown in Figure 3.14e. Each *div* contains either an *htab* or a *stab* whose visibility is toggled in the search functions. This procedure also creates the *iframes* that hold the individual panels.

```

</script>
</head><title>Search Demo
</title>
<body onLoad="showPanel(2);">

<table width=400><tr>
<td align=center nowrap>

<font size="4" face="verdana,arial,times" color="#000066"><strong>
CTD Laboratory Information Management System
</strong></td></tr></table>

<table align=center border=0 cellpadding=0 cellspacing=0 >
<tr>
<td>

<div id="htab2" class="tab" style="position: absolute; top:105; z-index:4;" onClick =
"showPanel(2);" onMouseOver="mouseOn('SEARCHOFF');window.status = 'Locate' "
onMouseOut="mouseOff('SEARCHOFF');window.status=''"><img src=SEARCHOFF.GIF"
title="Search" id="SEARCHOFF"></div>

<div id="htab3" class="tab" style="position: absolute; top:135 ; z-index:4;" onClick =
"showPanel(3);" onMouseOver="mouseOn('RESULTOFF');window.status = 'Query Results' "
onMouseOut="mouseOff('RESULTOFF');window.status=''"></div>

<div id="stab2" class="tab" style="position: absolute; top:105 ; z-index:4;
visibility:hidden; " onClick = "showPanel(2);" onMouseOver="window.status = 'Locate' "
onMouseOut="window.status=''"></div>

<div id="stab3" class="tab" style="position: absolute; top:135 ; z-index:4;
visibility:hidden; " onClick = "showPanel(3);" onMouseOver="window.status = 'Query
Results' " onMouseOut="window.status=''"></div>
</td>
</tr>
<tr>
<td>
</td>
</tr>');

http.p('</table>
<iframe id="panel2" class="panel" name="panel2" id="panel2" align="left" frameborder="0"
scrolling=no height="450"
src="SEARCH.display" ></iframe>

<iframe id="panel3" class="panel" name="panel3" id="panel3" align="left" frameborder="0"
scrolling=no height="450"
src="QUERYBLANK.DISPLAY"></iframe>
');

```

Figure 3.14e: LIMS Maintenance Records – Creating the Query Interface. This illustration shows the graphical methods used to create the query panels and buttons that control the search process.

The style section is used to present styles for the panels. This includes the size of the frames and position of the navigation buttons. The utility for organizing and displaying the elements is provided in Figure 3.14f. In this example, the analyst is provided complete control over the look and feel of the graphical user interface – including color implementation, position, and transparency of the various controls.

```

HTP.P('</body>
<style>
    .tab{
        color: clear;
        background-color: clear;
        border: clear;
        position: absolute;
        top: 12;
        left: 2;
        width: 3;
        text-align: center;
        font: 9pt Verdana,sans-serif;
        padding: 0;
        cursor: pointer;
        cursor: hand;
        scrolling: no;
    }
    .panel{
        position: absolute;
        align: left;
        top: 73;
        left: 0;
        width: 556;
        z-index: 1;
        visibility: hidden;
        font: 12pt Verdana,sans-serif;
        color: navy;
        border-style: none;
        margin: 0;
        padding: 0;
        overflow: none;
        frameborder: 1;
        border:0;
        hspace:0;
        vspace:0;
        scrolling: no;
    }
</style>

</BODY>
</HTML>');
end display;

PROCEDURE donothing
IS
BEGIN
    http.p('');
END donothing;

END search_start;
/

```

Figure 3.14f: LIMS Maintenance Records – Creating the Graphic User Interface. This illustration shows the graphical methods used to create the ancillary objects and query panels. The procedure *donothing* is used to create an empty string. This is required to initialize the *iframe* when no initial data is available.

Within Figures 3.14g-i, three procedures are shown that support the main search utility. Within Figure 3.14g, the *display* utility creates a frame search-box, which is populated with the procedure *display_detail*. The *display_detail* utility places a reset button at the top of the page, and creates an additional frame search-box. The button for this search is populated with the procedure *display_detail*. The *display_detail* utility creates the frame *searchpanel* that is further populated by *donothing* and displays a blank page. The *namesearch* utility creates the input box *textsearch* where the user enters the

string that will be searched. When the button search is pressed, the text in the box is passed to the package *find.display* algorithm.

```
CREATE OR REPLACE PACKAGE search AS
    procedure display;
    PROCEDURE display_detail;
    procedure namesearch;

END search;
/

CREATE OR REPLACE PACKAGE BODY search AS

    PROCEDURE DISPLAY
    IS
    BEGIN
        http.p(' <html><head></head>

<body bgcolor="white" link="#ffffff" vlink="#ffffff" alink="#ffffff"
topmargin="0" leftmargin="0" marginheight="0" rightmargin="0"
marginwidth="0">

<FORM name="aform">

<table align=left width=75 border=1 cellpadding=0 cellspacing=0 valign=top >
<tr><td bgcolor=#000066>


</td></tr>
</table>

<table bgcolor=#000066 width=860 align=left border=1>
<td align=left>

<iframe width=468 height=430 name="searchbox" id="searchbox" src="search.display_detail"
frameborder=0 scrolling=no></iframe>
</td>
</tr>

</table>
</FORM>
</body>
</html>' );

END;
```

Figure 3.14g: LIMS Maintenance Records – The Display Utility. In this example, the *display* utility creates a frame search-box that is further populated with the procedure *display_detail*. The suggestions are placed in the cascade sheets using *LoadCSS*.

The display utility operates with the display detail function to create the source image for the *iframe* search panel. This process is shown in Figure 3.14h, where the *iframe* is set to the parameter *search.namesearch*. This procedure places a reset button at the top of the page, and creates an additional frame search-box that is used to further develop the query. The algorithm is used to show (display) all the graphical elements that are required to support the query process and may be expanded to include additional controls or functions that allow the MOD user to enter various modes of operation. This procedure is commonly used in LIMS processing where specific details are displayed to analysts with prior levels of expertise. Hence, the Level I scientists are only shown specific controls and options that are commensurate with their skill level. Whereas, Level II engineers are shown the complete set of utilities that are required to support their advanced reporting standards.

```

PROCEDURE display_detail IS

BEGIN
  suggestions.LoadCSS;

  http.p(' <html><body><title></title><head><script>

function showSearch(){
document.getElementById('searchpanel').src='search.namesearch';
}

function reset_em(){
document.getElementById('searchpanel').src='search.namesearch';
}
</script>

</center>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000"
LINK="#FF0000" ALINK="#FF0000" VLINK="#FF0000"
topmargin="0" leftmargin="0"
marginheight="0" rightmargin="0"
marginwidth="0"
onLoad=showSearch();>

<TABLE WIDTH="440" BORDER="0" CELLPADDING="0" CELLSPACING="0" align="left">
  ');

  http.p('
<p><ol>
  <form name=aform onSubmit="false">
    <table border=0 width=100% columns=3>

<td class=fld align=center ></td>

<td class=fld align=center colspan=3 >
<input type="button" value="Reset"
onClick="reset_em();"></td>
  </tr><tr>

<td class=fld colspan=4 align=left>
<iframe frameborder=0 align=center width="440" height="340"
src="search_start.donothing"
name="searchpanel" id="searchpanel"
scrolling=yes marginheight="0" marginwidth="0">
  </iframe></td>');

  http.p('</tr>');
  http.print('</tr><tr>');
  http.print('</tr><tr>');

  http.print('</FORM>');

  http.print('</table>');

  http.print('</form>');
  http.print('</body>');
  http.print('</html>');

END;

```

Figure 3.14h: LIMS Maintenance Records – The *Display_Detail* Utility. The procedure places a reset button at the top of the page, and creates an additional frame search-box. The button for this search is populated with the procedure *display_detail*. The *onClick* methods are used to control the respective mouse operations.

```

procedure namesearch
is
begin
    suggestions.LoadCSS;
    http.p( '<html><body><head>
<script>

function reset_em(){
document.getElementById( 'searchpanel' ).src='search.namesearch';
}

function searchResults(srchname) {

parent.document.getElementById( 'searchpanel' ).src
='find.display?name='+srchname;

    }

</script></head>
<center>
<form name=aform onSubmit="return false">
<table border=1 width=100% >
<td class=fld align=center colspan=2 >
<input type=button name=search
value="Begin Search"
onClick="searchResults(document.aform.textsearch.value);"></td>

onClick="reset_em();"></td>-->
</table>
</td>

<td class=hfld>Search LIMS</td>
<td class=hfld><input type=text name=textsearch >

</td>

</table></center>
</form> </body></html> ' );

end;
END search;
/

```

Figure 3.14i: LIMS Maintenance Records – The *NameSearch* Utility. The procedure creates the input box *textsearch* where the user enters the string that will be searched. When the “Begin Search” control is activated, the JavaScript Function *searchResults* is called. This procedure sets the source of *iframe* search-panel to *find.display*.

The algorithms shown in Figure 3.14i perform the actual search requirements for the maintenance utility. The *display* procedure accepts a parameter name that is passed into the function from *namesearch*. This value is then appended to the SQL statement to select information where the *srchname* is the query for instrument name, *measurement_fcn* is the query for the measurement function, and, *fn* is the query for the item number.

Within Figure 3.14j, a hyperlink is used to record the result from each query. When the user clicks on the hyperlink, the panel *querydetail.display* is opened in the third hierarchical panel *panel3*, with the parameter *itemid* being set to the item# of the selected record. The function *showPanel(3)* is activated to transfer focus to *panel3*.

```

CREATE OR REPLACE PACKAGE find AS

PROCEDURE display(

name          in varchar2 default null);
END find;
/

CREATE OR REPLACE PACKAGE BODY find AS

PROCEDURE display(name          in varchar2 default null )
IS
stmt          varchar2(1000);
type          theRefCursor is Ref Cursor;
iCursor       thereofCursor;

type search_type is          table of varchar2(500);
item search_type;
instrument search_type;
measurement_fcn search_type;
fm search_type;
result varchar2(500);

BEGIN

suggestions.LoadCSS;

stmt:='select d.item,d.instrument,d.measurement_fcn,d.fm
from lab_equipment d where upper(item)
like  '%''||upper(:srchname) ||'%'' or

upper(instrument) like  '%''||upper(:srchname) ||'%''
or upper(measurement_fcn) like  '%''||upper(:srchname) ||'%''
or upper(fm) like  '%''||upper(:srchname) ||'%''';

open iCursor for stmt using name,name,name,name;

fetch iCursor bulk collect into
item,instrument,measurement_fcn,fm;

http.p('<table align=center border=0 width=100%>
<td align=left><b><u>Item</u></b></td><td
align=left><b><u>Instrument</u></b></td><td align=left><b><u>Measurement
Function</u></b></td></tr><tr>

');

```

Figure 3.14j: LIMS Maintenance Records – The Find and Display Utility (Part 1). The procedure controls the main search operations. Initially, a new table is created of type *varchar2* form. The main variables for instrument identification (*srchname*, *measurement_fcn*, and *fm*) are defined by *search_type*. These variables will be used to hold the values of information returned from the search query. The search statement queries table *lab_equipment* and searches for fields that match the search string. A “Like” clause is used so items that contain part of the search string will still be returned when the query is executed. The open *iCursor* function is used to execute the search string entered on the search form. The search string will be tested to see if it is similar to any of the fields in the table. The results of the query are then collected and stored in the data fields by instrument identification.

The query results are examined for validity. This includes the examination of null records when a query produces no results or an error has occurred within the LIMS or CTD materials database. If no records are returned, the search process is stopped and a JavaScript dialog box is presented to the user to notify them of the result. However, if the search produces valid records with matching strings, the results are displayed using the utilities shown in Figure 3.14k.

```

if item.count>0 then
for i in item.first..item.last
loop
if fm(i)= 'F' then
case(measurement_fcn(i))
when '1' then result:='Primary element detection and spectral analysis';
when '2' then result:='Tensile and compression testing';
when '3' then result:='Cyclic testing for tension, compression and material
ductility';
when '4' then result:='Primary hardness detection';
when '5' then result:='Optical measurement of metallographic specimen';
when '6' then result:='Sample preparation';
else result:=null;
end case;
else
case(measurement_fcn(i))
when '1' then result:='Primaryelement detection and spectral analysis';
when '2' then result:='NDT-UT (non-destructive testing, ultrasonic) flaw detection';
when '3' then result:='Network analysis using NDT-UT';
when '4' then result:='NDT-UT surface hardness';
when '5' then result:='NDT-UT material thickness';
when '6' then result:='Magnetic particle flaw detection';
when '7' then result:='Surface hardness (pin destructive testing)';
when '8' then result:='Optical measurement metallographic examination';
when '9' then result:='NDT residual stress analysis of sub-surface structure';
when '10' then result:='Pressure detection-vacuum detection';
when '11' then result:='Barometric pressure-hydrographic measurements';
when '12' then result:='Sample preparation and laboratory support';
when '13' then result:='Dynamic torque and acceleration';
when '14' then result:='Capillary and chemical amplification of surface features';
when '15' then result:='Physical resistance or conductance characterization';
when '16' then result:='Magnetic field detection and sub-element characterization';
else result:=null;
end case;
end if;
http.p('<td class=plainwhite><a target=panelbox2
href="QUERYDETAIL.DISPLAY?itemid='||item(i)||'"

onClick="parent.parent.parent.showPanel(3);">

<td class=plainwhite><a target=panelbox2
href="QUERYDETAIL.DISPLAY?itemid='||item(i)||'"

onClick="parent.parent.parent.showPanel(3);">

<!-- <td class=plainwhite><a target=panelbox2
href="QUERYDETAIL.DISPLAY?itemid='||item(i)||'"

onClick="parent.parent.parent.showPanel(3);">

-->

<td class=plainwhite><a target=panelbox2
href="QUERYDETAIL.DISPLAY?itemid='||item(i)||'"

onClick="parent.parent.parent.showPanel(3);">

<td class=plainwhite>--</td>
</tr><tr>');

end loop;
else
http.p('<script>alert("Your search for '||name||' yielded 0 results...Try Again");
parent.showSearch('name');</script>');
end if;
END;
END find;
/

```

Figure 3.14k: LIMS Maintenance Records – The Find and Display Utility (Part 2). The procedure controls the second phase of the search processing. In this example, instrument data is shown according to the simple layout provided in Figure 3.13a,b.

The process continues using the *querydetail.DISPLAY* procedure shown in Figure 3.14l. This algorithm queries the database for the *itemid* and returns all information recorded for that record. That information is then displayed in a separate frame.

```
CREATE OR REPLACE PACKAGE querydetail AS

    PROCEDURE DISPLAY(
        itemid          in varchar2 default null
    );
    PROCEDURE DISPLAY_GM;
END querydetail;
/

CREATE OR REPLACE PACKAGE BODY queryblank AS

    PROCEDURE display IS
    BEGIN
        suggestions.loadcss;
        http.p('<html><head></head>
<body bgcolor="white" link="#ffffff" vlink="#ffffff" alink="#ffffff"
topmargin="0" leftmargin="0" marginheight="0" rightmargin="0"
marginwidth="0">
<FORM name="aform">
<table align=left width=75 border=1 cellpadding=0 cellspacing=0 valign=top >
<tr><td bgcolor=#000066>

</td></tr>
</table>

<table bgcolor=#000066 width=468 align=left border=1><td align=left width=250>
<iframe width=110 height=430 name="panelbox" id="panelbox" src="queryblank.populate_title"
frameborder=0 scrolling=no></iframe>
</td>

<td align=left>
<iframe width=358 height=430 name="panelbox2" id="panelbox2" src="search_start.donothing"
frameborder=0 scrolling=yes></iframe>
</td>
</table>
</FORM>

</body>
</html>');

    END;

PROCEDURE POPULATE_TITLE
IS
BEGIN
    http.print('<table bgcolor=#FFFFFF><tr>
<td align=center>
<font size="3" face="verdana,arial,times" color="#0000cc"><strong>
CTD <br>Laboratory Information <br>Management System<br>
Query Results</strong></td></tr><tr><td><br><br></td></tr><tr>
<td align=center>
</td></tr></table>');
END;

END queryblank;
/
```

Figure 3.14l: LIMS Maintenance Records – The Find and Display Utility (Part 3). The procedure controls the third phase of the search process. The algorithm queries the CTD database by *itemid* and returns all information recorded for that record. That information is then displayed in a separate frame. The procedure *populate_title* is used to display a heading that appears in the left frame of the “Results” frame.

The procedure *Querydetail* displays additional information when the hyperlink returned by the initial query is selected. This *Display_gm* is a procedure that is used specifically for this case example-using

item 13.1. In this illustration a series of MOD data records are used to illustrate the setup for the query process. The case study is shown in Figure 3.14m,n. Within Figure 3.14m, the *querydetail* procedure is used setup the dataset. The *stmt := select* statement returns all information in the database for parameter *itemid*. The *fetch iCursor* statement is executed and bulk collected into the variables defined when *search_type2* was created. The function *openReport* opens the window name. This window is of size 550x260 and contains the options specified in the *window.open string*.

```
CREATE OR REPLACE PACKAGE querydetail AS

    PROCEDURE DISPLAY(
        itemid          in varchar2 default null
    );
    PROCEDURE DISPLAY_GM;
END querydetail;
/

CREATE OR REPLACE PACKAGE BODY querydetail AS

PROCEDURE DISPLAY (
    itemid          in varchar2 default null

) IS
    stmt                varchar2(1000);
    type mbrCursorType is ref cursor;

    iCursor mbrCursorType;
    equip_type varchar2(100);
    result varchar2(500);

    type search_type2 is    table of varchar2(4000);
    item1 search_type2;
    instrument search_type2;
    measurement_fcn search_type2;
    fm search_type2;
BEGIN
    suggestions.loadCSS;

    stmt:= 'SELECT item,instrument,measurement_fcn,fm FROM lab_equipment WHERE
item=:itemnumber ';
    iCursor FOR stmt using itemid;
    fetch iCursor bulk collect into item1,instrument,measurement_fcn,fm;
    http.p('<table align=left>

        ');
    http.print('
<HTML>
<HEAD>
<TITLE></TITLE>
<script>
function openReport(name){
    myWindow=window.open(name,"myWindow",

"toolbar=no,location=no,scrollbars=no,status=yes,resizable=no,location=no,toolbar=no,width
=550,height=260,top=10,left=10,screeny=25,screenx=50\"");
}
</script>
</HEAD> ');

    http.print('<table width=90 border=1 cellpadding=0 cellspacing=0 bgcolor=#FFFFFF
align=left>');
```

Figure 3.14m: LIMS Maintenance Records – The Find and Display Utility (Part 4). The procedure returns all information from the database for parameter *itemid*. A new window is created within the *openReport* function that contains attributes passed from the main CTD materials database.

The final elements of the database query are shown in Figure 3.14n,o. Each example uses data organized for the CTD laboratory system by instrument identification code. The parameters are

hardwired for this example to illustrate the process steps. In the main LIMS system, each parameter is passed back to this utility from the instrument identification codes and the metadata system records.

```

if item1.count>0 then
for i in item1.first..item1.last
loop
if fm(i)='F' then
equip_type:='Fixed Laboratory Equipment';
else
equip_type:='Mobil Laboratory Equipment';
end if;
if fm(i)= 'F' then
case(measurement_fcn(i))
when '1' then result:='Primary element detection and spectral analysis';
when '2' then result:='Tensile and compression testing';
when '3' then result:='Cyclic testing for tension, compression, material ductility';
when '4' then result:='Primary hardness detection';
when '5' then result:='Optical measurement of metallographic specimen';
when '6' then result:='Sample preparation';
else result:=null;
end case;
else
case(measurement_fcn(i))
when '1' then result:='Primaryelement detection and spectral analysis';
when '2' then result:='NDT-UT (non-destructive testing, ultrasonic) flaw detection';
when '3' then result:='Network analysis using NDT-UT';
when '4' then result:='NDT-UT surface hardness';
when '5' then result:='NDT-UT material thickness';
when '6' then result:='Magnetic particle flaw detection';
when '7' then result:='Surface hardness (pin destructive testing)';
when '8' then result:='Optical measurement metallographic examination';
when '9' then result:='NDT residual stress analysis of sub-surface structure';
when '10' then result:='Pressure detection-vacuum detection';
when '11' then result:='Barometric pressure-hydrographic measurements';
when '12' then result:='Sample preparation and laboratory support';
when '13' then result:='Dynamic torque and acceleration';
when '14' then result:='Capillary and chemical amplification of surface features';
when '15' then result:='Physical resistance or conductance characterization';
when '16' then result:='Magnetic field detection and sub-element characterization';
else result:=null;
end case;
end if;
http.p(' <tr><td class=lightgray align=left> Item:</td><td class=lightgray colspan=2
<tr><td class=lightgray align=left>Equipment Classification:</td><td
class=lightgray colspan=2 align=left>||equip_type||</td></tr>');
if item1(i) = '2.1' then
http.p(' <td class=lightgray align=left> Crack Propagation Test:</td><td class=lightgray
align=left>017358</td></tr>
<td class=lightgray align=left> Flaw Magnification Test:</td><td class=lightgray
align=left>152356</td></tr>
<td class=lightgray align=left> Subsurface Flaw Detection Test:</td><td class=lightgray
align=left>951753</td></tr>
<td class=lightgray align=left> Surface Structure Test:</td><td class=lightgray
align=left>456369</td></tr>
<td class=lightgray nowrap align=left> Flaw Detection Subsurface Element Test:</td><td
class=lightgray align=left>791385</td></tr>');
elseif item1(i) = 'F2.1' then
http.p(' <td class=lightgray align=left> Ductile Examination Test:</td><td class=lightgray
align=left>771395</td></tr>
<td class=lightgray nowrap align=left> Pre-examination for cycling Test:</td><td class=lightgray
align=left>113579</td></tr>');
elseif item1(i) = '13.1' then
http.p(' <td class=lightgray align=left>Pressure Dynamic Test:</td><td class=lightgray
align=left>34837443</td></tr>
<td class=lightgray align=left>Service Main Pressure Vessel:</td><td class=lightgray
align=left>78452111</td></tr>
<td class=lightgray align=left>Service Filter Elements:</td><td class=lightgray
align=left>65443331</td></tr>
<td class=lightgray align=left>Service Hydraulic Couplings:</td><td class=lightgray
align=left>87234526</td></tr>

```

Figure 3.14n: LIMS Maintenance Records – The Find and Display Utility (Part 5). Instrumentation and graphical layout parameters for the fixed and mobile laboratories.

```

<td class=lightgray nowrap align=left><a href="#"
    onClick="openReport('querydetail.DISPLAY_GM');"
    onMouseOver="status='Click To Display Additional Information About:
' || instrument(i) || '' ;return true"
    onMouseOut="status=' ' ;return true"
    title='Click To Display Additional Information About:
' || instrument(i) || '' >
    Lubrication and General Maintenance:</a></td>
    <td class=lightgray align=left><a href="#"
    onClick="openReport('querydetail.DISPLAY_GM');"
    onMouseOver="status='Click To Display Additional Information About:
' || instrument(i) || '' ;return true"
    onMouseOut="status=' ' ;return true"
    title='Click To Display Additional Information About:
' || instrument(i) || '' >
        84784788</a></td></tr>');
end if;
end loop;
end if;

http.print('</table>');
http.print('</form>');
http.print('</body>');
http.print('</html>');

END;

PROCEDURE DISPLAY_GM
IS
BEGIN
http.p('<html><body>
<table border=1>
<td align=center><b><u>Item</u></b></td>
<td align=center><b><u>Service Date</u></b></td>
<td align=center><b><u>Location</u></b></td>
<td align=center><b><u>Parts</u></b></td>
<td align=center><b><u>Engineer</u></b></td>
</tr><tr>
<td align=left>Main Pinion Bearing</td>
<td align=left>2 FEB 2004</td>
<td align=left>CTD Fixed Lab</td>
<td align=left>34-3097-51</td>
<td align=left>M.Gobelev</td>
</tr><tr>
<td align=left>Belt Assembly</td>
<td align=left> 11 FEB 2004 </td>
<td align=left>CTD Mobile Lab</td>
<td align=left>45-4098-12 </td>
<td align=left>H. Stubovin</td>
</tr><tr>
<td align=left>Fan Housing </td>
<td align=left>1 MAR 2004 </td>
<td align=left>Ishgorskij Lab </td>
<td align=left>456-551-234 </td>
<td align=left>A. Demiskij</td>
</tr><tr>
<td align=left>Brass Couplings</td>
<td align=left>12 MAR 2004 </td>
<td align=left>Railway Lab </td>
<td align=left>339-119-02 </td>
<td align=left> A. Feleskov</td>
</table>
</html></body>');
END;

END querydetail;
/

```

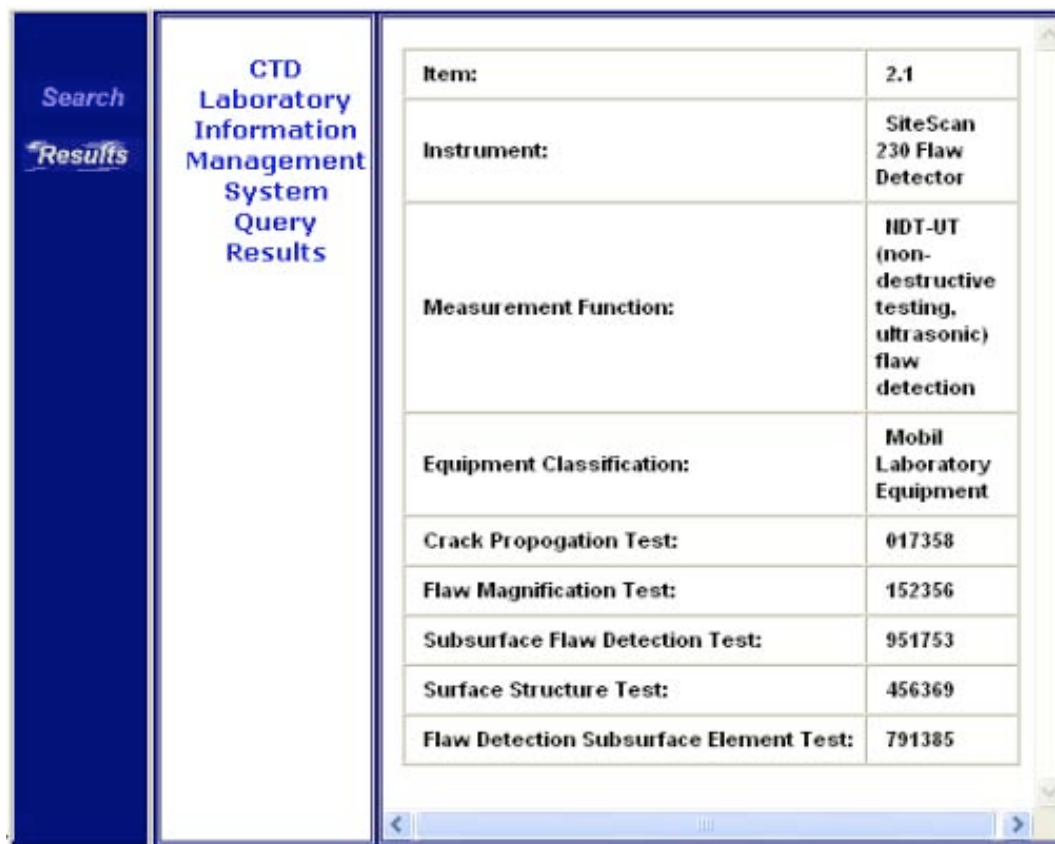
Figure 3.14o: LIMS Maintenance Records – The Find and Display Utility (Part 6). Instrumentation and graphical layout parameters for the fixed and mobile laboratories – final elements.

The algorithms shown in Figure 3.14b-o are used to support the CTD maintenance and search utility that operates within the fixed and mobile laboratory system. The results of this application are shown in Figure 3.15a-d. Within Figure 3.15a, the initial search shown in Figure 3.14a has been completed and a single instrument located that contains the SiteScan name. The search results include three hyperlinks.



<u>Item Instrument</u>	<u>Measurement Function</u>
<u>2.1 SiteScan 230 Flaw Detector</u>	<u>NDT-UT (non-destructive testing, ultrasonic) flaw detection</u>

Figure 3.15a: LIMS Maintenance Records – Searching MOD Archives for Sitescan Instrumentation. In this example, the search described in Figure 3.14a has been conducted using the algorithms shown in Figure 3.14b-o. In this example, one instrument type (SiteScan 230 Flaw Detector) has been located by measurement function.



Item:	2.1
Instrument:	SiteScan 230 Flaw Detector
Measurement Function:	NDT-UT (non-destructive testing, ultrasonic) flaw detection
Equipment Classification:	Mobil Laboratory Equipment
Crack Propagation Test:	017358
Flaw Magnification Test:	152356
Subsurface Flaw Detection Test:	951753
Surface Structure Test:	456369
Flaw Detection Subsurface Element Test:	791385

Figure 3.15b: LIMS Maintenance Records – Querying the MOD Archive for Sitescan Flaw Detection Tests. In this example, the search returned five tests with specific identification codes that resulted from the LIMS QA/QC procedures.

Each link is oriented by: Item number, Instrument Type, and Measurement Function. The analyst then selects the hyperlink to display additional information of the form shown in Figure 3.15b. In this example, new information is provided concerning the identification codes for each respective test that was conducted using the SiteScan 230 Flaw Detection System. The identification codes can be indexed back to the main LIMS system or the CTD/Oracle materials Database.



Search
Results

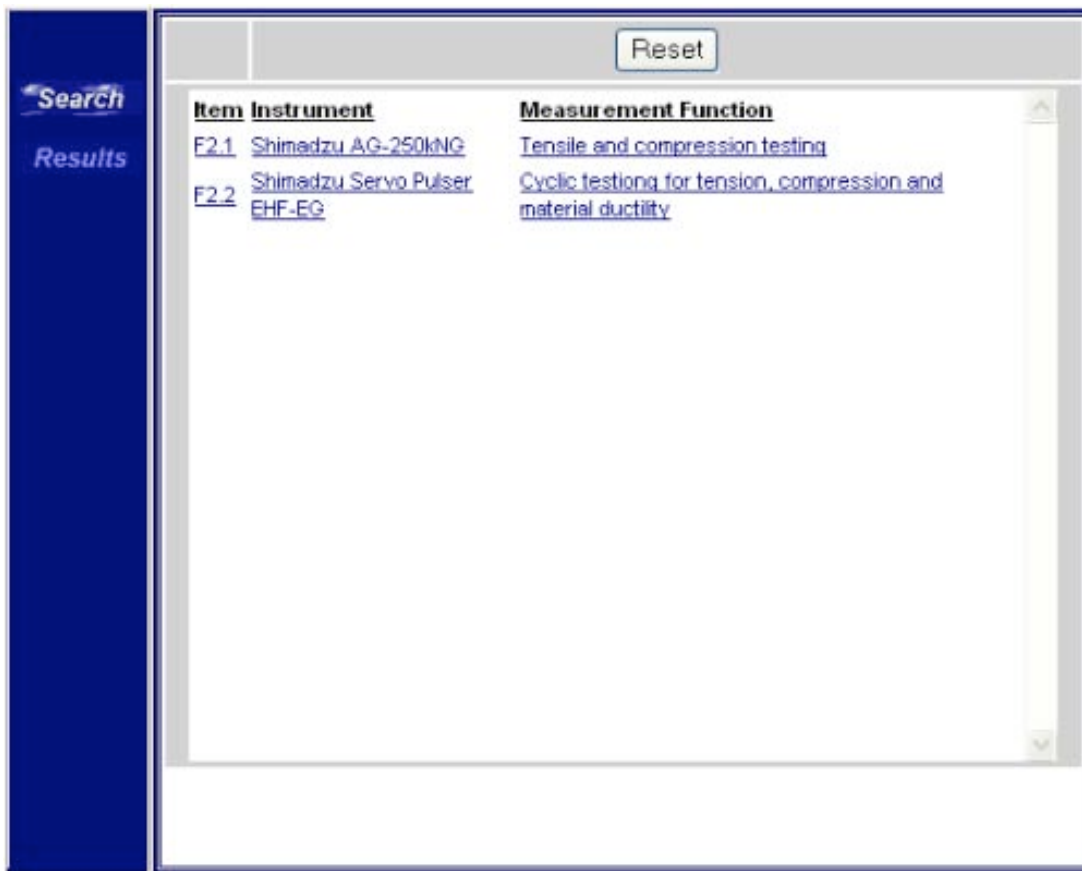
Reset

Begin Search

Search LIMS

Shima

Figure 3.15c: LIMS Maintenance Records – Querying the MOD Archive for Shimadzu Destructive Testing Systems.



Item	Instrument	Measurement Function
F2.1	Shimadzu AG-250kNG	Tensile and compression testing
F2.2	Shimadzu Servo Pulser EHF-EG	Cyclic testing for tension, compression and material ductility

Figure 3.15d: LIMS Maintenance Records – Search Results for Shimadzu Destructive Testing Systems. The query shown in Figure 3.15c has resulted in two systems: F2.1 and F2.2 corresponding to the Shimadzu AG-250kNG and the Shimadzu Servo Pulser EHF-EG.

As shown in Figure 3.15c, the initial query may be redirected toward other instruments located in the CTD fixed or mobile laboratory system. In this example, a query is starting for any system that contains the phrase “Shima”. The function will context-match the remaining portions of the search string and return all Shimadzu related tests that have been conducted by MOD. In this example, only

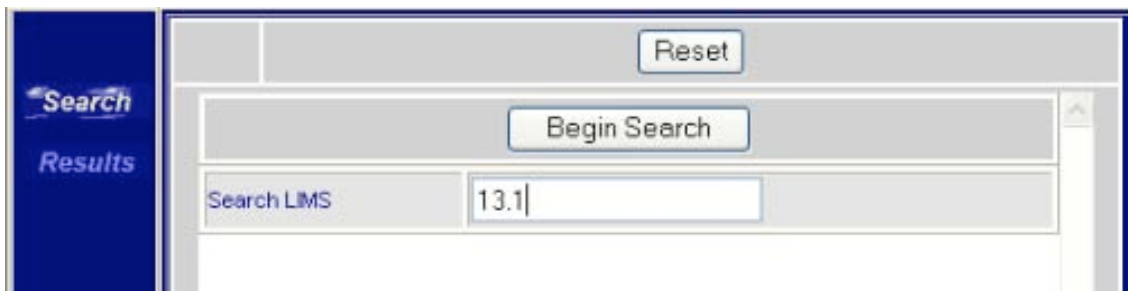
Search

Results

CTD
Laboratory
Information
Management
System
Query
Results

Item:	F2.1
Instrument:	Shimadzu AG-250kNG
Measurement Function:	Tensile and compression testing
Equipment Classification:	Fixed Laboratory Equipment
Ductile Examination Test:	771395
Tensile Examination Test:	956871
Compression Examination Test:	338412
Combined Examination Test:	828463
Pre-examination for cycling Test:	113579

Figure 3.15e: LIMS Maintenance Records – Hyperlink Data from the Shimadzu AG-250kNG Destructive Testing System. The search indicates that five specific tests have been conducted on this instrument. Each test is assigned a unique identification code from the main LIMS system.



Search Results	<table><tr><td><input type="button" value="Reset"/></td></tr><tr><td><input type="button" value="Begin Search"/></td></tr><tr><td><input type="text" value="13.1"/></td></tr></table>	<input type="button" value="Reset"/>	<input type="button" value="Begin Search"/>	<input type="text" value="13.1"/>
<input type="button" value="Reset"/>				
<input type="button" value="Begin Search"/>				
<input type="text" value="13.1"/>				

Figure 3.15f: LIMS Maintenance Records – Search Results by Equipment Item. In this example, the user is conducting a query by item number as opposed to equipment name or function.

two instruments match the search context: the Shimadzu AG-250kNG and the Shimadzu Servo Pulser EHF-EG. Each system contains a series of hyperlinks to extract additional data shown in Figure 3.15e,f. The search returns five specific tests that have been conducted using the AG-250kNG. The identification codes have been assigned to each test sequence from within the LIMS, and may be further examined from within this search utility or from within the main StarLIMS shell. Within Figure 3.15g, a query is shown for the item number or identification number of a specific LIMS instrument. In this case, the item 13.1 refers to the Start1M Compressor system that is used in the fixed laboratory (destructive system). As shown in this example, the query has revealed specific



<u>Item</u>	<u>Instrument</u>	<u>Measurement Function</u>
<u>13.1</u>	<u>Start 1M Compressor</u>	<u>Sample preparation and laboratory support</u>

Figure 3.15g: LIMS Maintenance Records – Search Results by Equipment Item – Start 1M Compressor. In this example, the query “13.1” has returned a data record (hyperlink) to the Start 1M Compressor system located in the fixed laboratory. The record indicates that sample preparation and laboratory support functions are approved for this system.

information concerning sample preparation and laboratory support functions. Detailed information for the Start 1M Compressor is shown in Figure 3.15h. In this example, the user has selected the hyperlink for sample preparation and laboratory support. This displays the results shown in this illustration. The records indicate that certain tests have been performed as well as specific maintenance. In particular, the system has been examined for Lubrication and General Maintenance yielding a service record (unique identification code) equal to 84784788. As shown, the analyst can select the hyperlink to display additional information about the maintenance records for the Start 1M Compressor.

CTD Laboratory Information Management System

Search

Results

CTD
Laboratory
Information
Management
System
Query
Results

Item:	13.1
Instrument:	Start 1M Compressor
Measurement Functions:	Sample preparation and laboratory support
Equipment Classification:	Mobil Laboratory Equipment
Pressure Dynamic Test:	34837443
Service Main Pressure Vessel:	78452111
Service Filter Elements:	65443331
Service Hydraulic Couplings:	87234526
Lubrication and General Maintenance:	84784788

Click To Display Additional Information About: Start 1M
Compressor

Click To Display Additional Information About: Start 1M
Compressor

Figure 3.15h: LIMS Service and Maintenance Records for the Start 1M Compressor. This query indicated specific tests and service and support functions that have been conducted on this system. By moving the cursor over the hyperlink, a pop-up window is shown that directs the user to additional information from the LIMS and CTD Materials Database.

When the user selects the hyperlink for Lubrication and General Maintenance additional information is provided concerning the specific service records for the device. The records are shown in Figure 3.15i organized by service date (and service engineer) that conducted the examination. The database also includes records for the main parts that have been replaced or repaired.

<u>Item</u>	<u>Service Date</u>	<u>Location</u>	<u>Parts</u>	<u>Engineer</u>
Main Pinion Bearing	2 FEB 2004	CTD Fixed Lab	34-3097-51	M.Gobelev
Belt Assembly	11 FEB 2004	CTD Mobile Lab	45-4098-12	H. Stubovin
Fan Housing	1 MAR 2004	Ishgorskij Lab	456-551-234	A. Demiskij
Brass Couplings	12 MAR 2004	Railway Lab	339-119-02	A. Feleskov

Done	Internet
Service Hydraulic Couplings:	87234526
<u>Lubrication and General Maintenance:</u>	<u>84784788</u>

Figure 3.15i: LIMS Service and Maintenance Records for the Start 1M Compressor (Continued). The hyperlink for Lubrication and Maintenance has resulted in four main service events. These include service or repair on the Main Pinion Bearing, Belt Assembly, Fan Housing, and Brass Couplings. Each service event includes the part numbers for the repaired item and the name of the field service engineer that conducted the examination.

4.0 Certification and Attestation Programs

The Applied Logic Laboratory has completed the Level II attestation training for three members of the MOD 12th Main Directorate: Andrew Vladimirovich Shishov, Denis Alexandrovich Chivilev, and Juriy Sergeevich Afanasyev. This included the recertification of the students for all mobile operations at the Level I standard and re-attestation of the students for Level II with comprehensive examinations from Gosgorteknadzor.

At the present time, six additional students are undergoing training for the safety and service-life characterization of: cranes, hoists, and mobile lifting systems (including portable pneumatic and hydraulic assemblies). These students include: Oleg Stanislavovich Ratushny, Alexey Yurievich Boiko, Vitaliy Victorovich Manulenko, Paul Pavlovich Igolaynen, Victor Vladimirovich Kakurin, and Ivan Alexeevich Smirnov (Figure 4.1). The new attestation training will be completed in 2004 based upon the examination schedule and the abilities of each student to receive the Level II certification. Since certain students do not have the necessary technical prerequisites for this level of certification, ALL is also completing the technical training that is required for their promotion to the Level II technical standard according to Gosgorteknadzor.



Figure 4.1: Student Instruction and Technical Examination. MOD 12th Main Directorate working on practical laboratory examinations. The practical exams are used to prepare the students for the formal examination sequence from Gosgorteknadzor.

Within this discussion, we provide the Gosgorteknadzor examination and attestation documents for the new students that have successfully completed their examinations. The documentation includes the Level II examinations with the individual test scores for each of the respective students by curriculum. Photographs are provided that show each student receiving their diplomas from the Gosgorteknadzor representative in Moscow. The documentation also includes the Level I and Level II attestation certificates that allow each student to: perform field diagnostics and report technical findings to the main laboratory. The student examination process is documented in Figure 4.2a to Figure 4.8c.



Figure 4.2a: Level II Examinations and Technical Diploma for Yuriy Sergeevich Afanasyev.



Figure 4.2b: Gosortekhnadzor Attestation for Yuriy Sergeevich Afanasyev.

Прошел аттестацию в качестве специалиста в области экспертизы промышленной безопасности подъемных сооружений.

Уровень квалификации и область аттестации												
7	8	9	10	11	12	13	14	15	16	17	18	19
2	2	2	2	2	2	2	2	2	2	2	2	2

Протокол №5-Э/04
Дата 30.01.2004г.

Срок действия
30.01.2007г.

М.П. [Signature]

М.П. [Signature]



Figure 4.2c: Level II Examinations for Yuriy Sergeevich Afanasyev. Practical examinations are shown with the resultant certification levels for each testing sequence. An "x" indicates that the student did not pass the examination based upon a technical deficiency. A "1" indicates the Level I Attestation. A "2" indicates that the student has passed the examination at the Level II Gogorteknazdor Attestation.



Figure 4.3a: Level II Examinations and Technical Diploma for Andrew Vladimirovich Shishov.



Figure 4.3b: Gosortekhnadzor Attestation for Andrew Vladimirovich Shishov.

Прошел аттестацию в качестве специалиста в области
экспертизы промышленной безопасности подъемных
сооружений.

Уровень квалификации и область аттестации												
А-1	А-2	А-3	А-4	А-5	А-6	А-7	А-8	А-9	А-10	А-11	А-12	А-13
								x	2	x	x	2

Протокол №5-Э/04
Дата 30.01.2004г.

Срок действия
30.01.2007г.

М.П. *[Signature]*

М.П. *[Signature]*

М.П. *[Signature]*

Figure 4.3c: Level II Examinations for Andrew Vladimirovich Shishov. Practical examinations are shown with the resultant certification levels for each testing sequence. An "x" indicates that the student did not pass the examination based upon a technical deficiency. A "1" indicates the Level I Attestation. A "2" indicates that the student has passed the examination at the Level II Gorteknazdor Attestation.



Figure 4.4a: Level II Examinations and Technical Diploma for Denis Alexandrovich Chivilev.

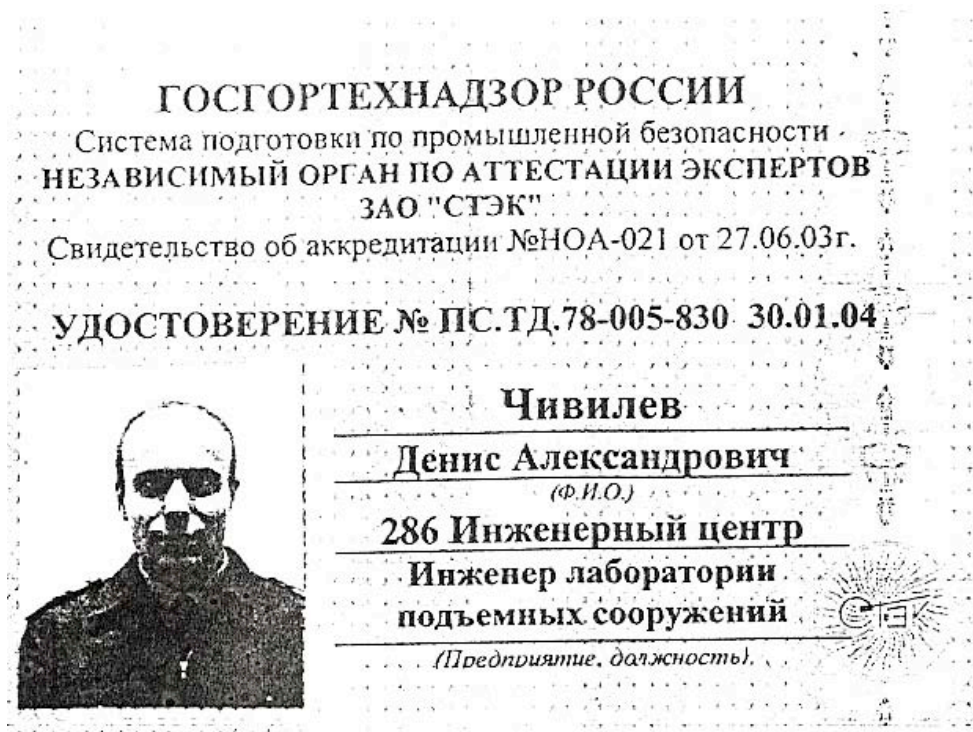


Figure 4.4b: Gosortekhnadzor Attestation for Denis Alexandrovich Chivilev.

Прошел аттестацию в качестве специалиста в области
экспертизы промышленной безопасности подъемных
сооружений

Уровень квалификации и область аттестации		Сектор						
		А-8	А-9	А-10	А-11	А-12	А-13	А-14
Инженер	2	x	2	x	x	x	x	1

Протокол №5-Э/04
Дата 30.01.2004г.

Срок действия
30.01.2007г.

ПЕТЕРБУРГСКАЯ
ТЕХНИЧЕСКАЯ
ЭКСПЕРТНАЯ
КОМПАНИЯ
Председатель
Сектор

М.П.

М.П.

Копия верна:

Q

Figure 4.4c: Level II Examinations for Denis Alexandrovich Chivilev. Practical examinations are shown with the resultant certification levels for each testing sequence. An "x" indicates that the student did not pass the examination based upon a technical deficiency. A "1" indicates the Level I Attestation. A "2" indicates that the student has passed the examination at the Level II Gogorteknazdor Attestation.

ВЫПИСКА ИЗ ПРОТОКОЛА № 5-Э/04 от 30 января 2004 г.

заседания аттестационной комиссии ЗАО "СТЭК" по проверке Правил Госгортехнадзора России и другой нормативной документации у специалистов по обследованию и техническому диагностированию грузоподъемных машин.

г. Санкт-Петербург

"30" января 2004 г.

Наименование курса: Обследование грузоподъемных кранов и подъемников (вышск).

Председатель: Бардышев О. А., директор ЗАО "СТЭК" по экспертной и учебной работе, руководитель НОА-021, эксперт.
Члены комиссии: Уралов В. Л., к.т.н., профессор, специалист-обследователь, эксперт.
Хроленко Ю. Г., начальник отдела обследования подъемных сооружений ЗАО "СТЭК", III уровень.
Представитель ГТН: Рыбников Г. В., начальник отдела по надзору за г/п кранами УСЗО ГТН.

Свидетельство об аккредитации № НОА-021

Проводилась проверка знаний: требований промышленной безопасности при эксплуатации опасных производственных объектов, РД 10-528-03, 10-382-00, ПБ-10-611-03; СНиП 12-03-01; ИСО 4308/1; РД-10-112-96, части 1-5, 8, 9; РД 10-138-97, РДИ-10-349(138)-00, РД-10-117-95, ПБ 03-246-98, ГОСТ Р 51248-99, ПБ-10-257-98, ПБ-10-157-97, ПБИ 10-371(157)-00, ПБ 10-518-02, ПБ 10-559-03.

Результаты аттестации:

Данные о слушателях курсов					Уровень квалификации и область аттестации												Заключение комиссии		
№ п/п	№ удост.	Фамилия, имя, отчество	Занимаемая должность	Место работы	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12		A13	A14
37.	78-005-830	Чивилев Денис Александрович	Инженер лабораторий подъемных сооружений	286 Инженерный центр	2	2	2	1	1	X	2	2	X	2	X	X	X	1	Выдать удост. на 3 года
38.	78-005-831	Шишов Андрей Владимирович	Старший инженер лабораторий подъемных сооружений	286 Инженерный центр	2	2	2	1	1	X	2	2	X	2	X	X	X	2	Выдать удост. на 3 года

Figure 4.5a: Level II Curriculum Attestation from Gosgorteknadzor (Side 1).

Figure 4.5b: Level II Curriculum Attestation from Gosgorteknadzor (Side 2).

**Аттестационный центр
«НИКИМТ»**

Свидетельство об аккредитации № 24209 от 14.03.2003 г.
Срок действия до 14.03.2006 г.

**Квалификационное удостоверение
№ 09-086-04**

Фамилия **Афанасьев**
Имя **Юрий**
Отчество **Сергеевич**
Год рождения **1977**



(подпись владельца)

(подпись руководителя)



Figure 4.6a: Russian State Certification for Yuriy Sergeevich Afanasyev. Technical diploma awarded on 4 February 2004 for field investigation in technical lifting systems, hoists, control systems and high pressure piping and boiler investigation.

Квалификационное удостоверение № 09-086-04

Уровень квалификации, вид (метод) контроля, наименование (индекс) объектов контроля в соответствии с Правилами аттестации персонала в области неразрушающего контроля.

Настоящее удостоверение действительно только при наличии удостоверения о проверке знаний правил безопасности.

Вид контроля	УК		ВИК		ПВК	
	мес.	год	мес.	год	мес.	год
Уровень 1						
Оборуд.						
2	02	2007	02	2007		
Оборуд.	3		3			
3						
Оборуд.						

(Подпись руководителя)

М.П. Адрес Независимого органа : **04.02.2004**
127410, Москва, Алтуфьевское ш., 43 (Дата выдачи)



Figure 4.6b: Russian State Certification for Yuriy Sergeevich Afanasyev. The certificate of certification from Gosgortekhnadzor to perform field diagnostics and technical evaluation in lifting and piping systems.

АТТЕСТАЦИОННЫЙ ЦЕНТР «НИКИМТ»**ИТОГОВЫЙ ПРОТОКОЛ ЭКЗАМЕНОВ**№ 162-04 от «04» ноября 2004 г.по проверке знаний по Ультразвуковомуметоду контроля на II уровень

Фамилия Имя Отчество	Образование	Место работы, должность, стаж работы	Вид проверки (очередная, первичная)
<u>Афанасьев</u>	<u>высшее</u>	<u>ЗСБ ИСТД,</u>	<u>очередная</u>
<u>Юрий</u>		<u>инженер,</u>	
<u>Сергеевич</u>		<u>4 года</u>	

Оценка по общему экзамену — %Оценка по специальному экзамену 95 %Оценка по практическому экзамену 95 %Итоговая оценка 95 %Результат экзамена: Афанасьеву Ю.С. присвоен II уровень
по Ультразвуковому методу контроляОборудования и трубопроводов, подконтрольных Госгортехнадзору РФ
п. 3

Приложения 1 «Правил аттестации персонала в области НК»

Директор Аттестационного
центраА.А.ПлаксинПредседатель экзаменационной
комиссииА.А.Плаксин

Figure 4.6c: Technical examination Yuriy Sergeevich Afanasyev. The student has completed the prescribed training with the scores shown in this figure (95%). The attestation level for this examination is grade II field diagnostics.



Figure 4.7a: Russian State Certification for Andrew Vladimirovich Shishov. Technical diploma awarded on 4 February 2004 for field investigation in technical lifting systems, hoists, control systems and high pressure piping and boiler investigation.

Квалификационное удостоверение № 09-085-04

Уровень квалификации, вид (метод) контроля, наименование (индекс) объектов контроля в соответствии с Правилами аттестации персонала в области неразрушающего контроля.

Настоящее удостоверение действительно только при наличии удостоверения о проверке знаний правил безопасности.

Вид контроля	УК		ВИК		ПВК	
	мес.	год	мес.	год	мес.	год
1						
Оборуд.						
2	02	2007	02	2007		
Оборуд.	3		3			
3						
Оборуд.						

(Подпись руководителя) М.П. Адрес Независимого органа: **04.02.2004**
127410, Москва, Алтуфьевское ш., 43 (Дата выдачи)

Figure 4.7b: Russian State Certification for Andrew Vladimirovich Shishov. The certificate of certification from Gosgorteknadzor to perform field diagnostics and technical evaluation in lifting and piping systems.

АТТЕСТАЦИОННЫЙ ЦЕНТР «НИКИМТ»**ИТОГОВЫЙ ПРОТОКОЛ ЭКЗАМЕНОВ**№ 111-04 от «04» декабря 2004 г.по проверке знаний по компьютерномуметоду контроля на II уровень

Фамилия Имя Отчество	Образование	Место работы, должность, стаж работы	Вид проверки (очередная, первичная)
<u>Шишов</u>	<u>высшее</u>	<u>206 ИСТД</u>	<u>очередная</u>
<u>Андрей</u>		<u>старший инженер</u>	
<u>Владимирович</u>		<u>Нидр</u>	

Оценка по общему экзамену — %Оценка по специальному экзамену 92,5 %Оценка по практическому экзамену 90 %Итоговая оценка 90,9 %Результат экзамена: Шишову АВ присвоен II уровень
по компьютерному методу контроляОборудования и трубопроводов, подконтрольных Госгортехнадзору РФ
п. 3

Приложения 1 «Правил аттестации персонала в области НК»

Директор Аттестационного
центраА.А.ПлаксинПредседатель экзаменационной
комиссииГенерал И.И.Реденко

Figure 4.7c: Technical examination Andrew Vladimirovich Shishov. The student has completed the prescribed training with the scores shown in this figure (90% - 92.5%). The attestation level for this examination is grade II field diagnostics.

**Аттестационный центр
«НИКИМТ»**

Свидетельство об аккредитации № 24209 от 14.03.2003 г.
Срок действия до 14.03.2006 г.

**Квалификационное удостоверение
№ 09-087-04**

Фамилия **Чивилев**
Имя **Денис**
Отчество **Александрович**
Год рождения **1976**





(подпись владельца)


(подпись руководителя)

Figure 4.8a: Russian State Certification for Denis Alexandrovich Chivilev. Technical diploma awarded on 4 February 2004 for field investigation in technical lifting systems, hoists, control systems and high pressure piping and boiler investigation.

Квалификационное удостоверение № 09-087-04

Уровень квалификации, вид (метод) контроля, наименование (индекс) объектов контроля в соответствии с Правилами аттестации персонала в области неразрушающего контроля.
Настоящее удостоверение действительно только при наличии удостоверения о проверке знаний правил безопасности.

Вид контроля	УК		ВИК		ПВК			
	мес.	год	мес.	год	мес.	год	мес.	год
1								
Оборуд.								
2	02	2007	02	2007				
Оборуд.	3		3					
3								
Оборуд.								

 М.П. Адрес Независимого органа. **04.02.2004**
(Подпись руководителя) 127410, Москва, Алтуфьевское ш., 43 (Дата выдачи)



Figure 4.8b: Russian State Certification for Denis Alexandrovich Chivilev. The certificate of certification from Gosgorteknadzor to perform field diagnostics and technical evaluation in lifting and piping systems.

АТТЕСТАЦИОННЫЙ ЦЕНТР «НИКИМТ»**ИТОГОВЫЙ ПРОТОКОЛ ЭКЗАМЕНОВ**№ 115-04 от «04» февраля 2004 г.по проверке знаний по Ультразвуковомуметоду контроля на II уровень

Фамилия Имя Отчество	Образование	Место работы, должность, стаж работы	Вид проверки (очередная, первичная)
<u>Чивилев</u>	<u>Высшее</u>	<u>Зав. ЦМТД</u>	<u>очередная</u>
<u>Денис</u>		<u>инженер</u>	
<u>Александрович</u>		<u>4 года</u>	

Оценка по общему экзамену — %Оценка по специальному экзамену 95 %Оценка по практическому экзамену 95 %Итоговая оценка 95 %

Результат экзамена: Чивилев Д.А. присвоен II уровень
по ультразвуковому методу контроля

Оборудования и трубопроводов, подконтрольных Госгортехнадзору РФ
п. 3

Приложения 1 «Правил аттестации персонала в области НК»

Директор Аттестационного
центра А.А.Плаксин

Председатель экзаменационной
комиссии А.А.Плаксин



Figure 4.8c: Technical examination Denis Alexandrovich Chivilev. The student has completed the prescribed training with the scores shown in this figure (95%). The attestation level for this examination is grade II field diagnostics.

5.0 Level II Field Diagnostics and Attestation Training

During the period December 2003 to March 2004, field diagnostic training was conducted for the MOD 12th Main Directorate by the faculty and staff from the St. Petersburg Technical Experts Company, The St. Petersburg Railway Institute, the St. Petersburg Geophysical Institute, and the Ishgorskij Technical Institute. The training was designed to support the MOD field operations using the major ultrasonic testing instruments from Sonatest and Stresstel and well as the support training that is required for MOD to retain their Level I Certifications from RSS and Gosgorteknadzor. As shown in Figure 5.1, the training was conducted within a fixed laboratory setting for initial certification. The students work with the RSS instructor to fully understand the instrumentation using standardized sample (calibration standards). In this example, the MOD student is working with the Stresstel T-Mike system to determine thickness characteristics as a component of an overall field hardness evaluation.



Figure 5.1: Laboratory Certification and Instrumentation Using the StressTel T-Mike EL System. The student receives one-to-one instruction with technical documentation from the manufacturer. The instruction is designed to assist the student in data analysis and data migration methods for the LIMS.

As shown in Figure 5.2, the student received detailed instruction for methods required to monitor corrosion and field deformation that will be modeled within the LIMS. This includes sample

techniques for identifying the stress-risers in the sample and comparing actual field observations with a priori samples (test-blocks) that have been prepared by RSS and Gosgorteknadzor. In this illustration, the student uses the StressTel T-Mike EL system to determine localized hardness characteristics in conjunction with field rated standards for fatigue (service manual) acquired for the lifting manufacturer. This allows the student to compare actual field cycles against certified capacities from the crane, hoist, or boiler manufacturer.



Figure 5.2: Fatigue Characteristics and Model Service Life Determination. In this laboratory exercise, the student works with the RSS instructor to prepare a contrast model for estimating fatigue (duty strength) in a field sample. A contrast model is created using the test-block as a standardized specimen. The loading factors and approximate fatigue characteristics are shown in the service manual (blue diagrams on the workbench surface).

The methods for characterizing a surface structure are examined by the MOD student. Repeated samples are acquired across the planar surface and data is registered by surface position. As shown in Figure 5.3a, the student initially works with standardized surfaces that contain certain imperfections. Note that this test block includes some pitting and surface rust (oxidation) that significantly influence the test results (final determination). Following the data acquisition phase, modeling is used to remove sample outliers and further identify probable locations for examination. In Figure 5.3b, a series of test samples are examined for fatigue and service characterization. Notice these samples are



Figure 5.3a: Surface Characterization with Standardized Samples and test Blocks.



Figure 5.3b: Surface Contrast Analysis Using Chemical and Capillary Techniques.

sequentially treated with chemicals (capillary determination) to identify stress-risers and probable sites for further analysis. The chemical methods are also used to identify (amplify) cracks or localized imperfections that may be present in the sample. The amplified sections are then examined using the Sonatest, Stresstel (or equivalent) ultrasonic defectoscopes. An example of this process is shown in Figure 5.4 using the Russian defectoscopes from the Ishgorskij Technical Institute.



Figure 5.4: Laboratory Procedures for Ultrasonic Identification. In this illustration, the MOD students prepare a laboratory diagnostic report for the test sample using the Russian (Dubeliv) Ultrasonic Defectoscope (Sonatest, Stresstel equivalent). The student is moving a transducer across the test surface as a method to identify internal stresses, cracks, moment deformations, and imperfections. The system requires each student to understand the instrumentation and the proper evaluation techniques for technical reporting and fatigue life estimation.

Within Figure 5.5a-b, the students continue their training using coupled transducers to detect surface fractures and a priori known deformations. In Figure 5.5a, the test block includes pre-drilled holes that are equally spaced along a known transect. The student uses the defectoscope to identify the location of each hole and register the specific coordinates. This testing sequence is commonly used during formal examinations, where the sample holes are not visible to the student. The examination is then setup in a blind manner where the student must identify all known deformations, surface fractures, and geometric conditions (drilled holes) without visually seeing these deformations. Hence, the logical movement of the transducer and the proper registration of the instrument are the key features for the successful determination.



Figure 5.5a: Ultrasonic Flaw Detection. Surface Methods Using Coupled Transducers.



Figure 5.5b: Ultrasonic Flaw Analysis – Sweep Detection Methods.



Figure 5.6a: Crane and Hoist Service Life Determination.

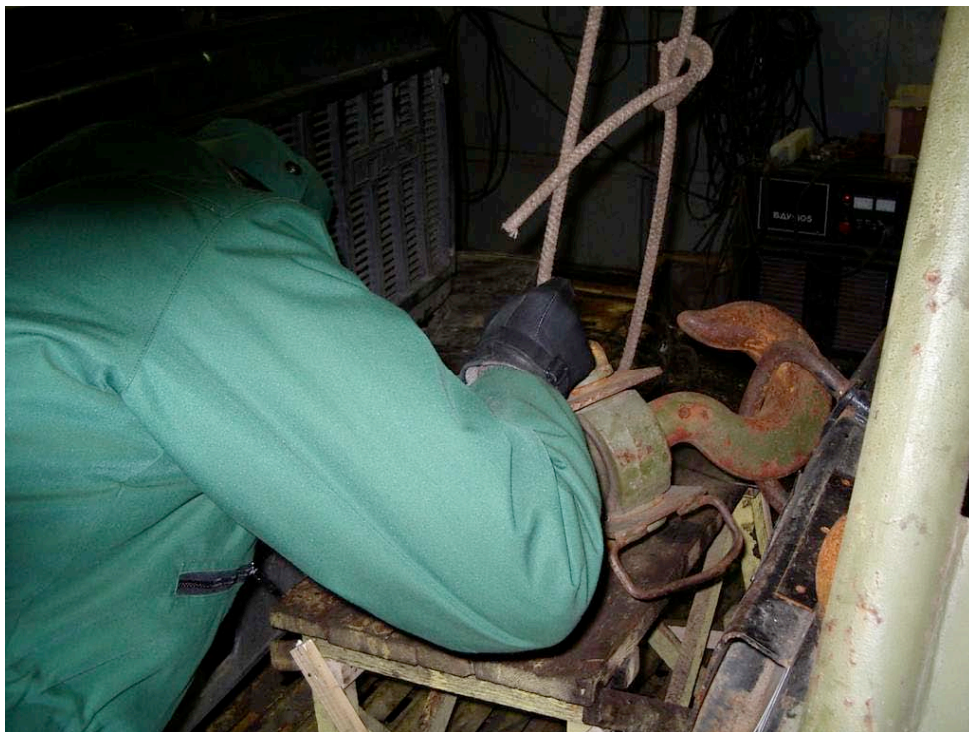


Figure 5.6b: Characterizing the Surface Condition of the Hoist Mechanism.

The students receive formal instruction in methods for service life determination (Figure 5.7). At the present time, only the students that have received their Level II Gosgorteknadzor attestation may recommend finding for determining the service life on hosts, cranes, elevator systems, and high pressure piping or boiler systems. While these students may recommend a service life condition, their reports must be reviewed by a Level III official that records the final information. As shown in Figure 5.6a-b, the characterization requires each student to examine the principal working mechanisms including the cable assemblies and the attachment hoists (or turnbuckle assemblies). The students have received training in magnetic cable deformation methods using the Intros systems that were supplied to the MOD field teams by DTRA.



Figure 5.7: Crane Examination Techniques. In this figure, the MOD student is moving along a truss section examining the surface conditions for individual welded sections. The student is also applying capillary methods to chemically amplify the surface deformation. In this manner, the principal cracks and surface deformations may be identified and marked for secondary examination using the surface hardness testing instruments and the ultrasonic defectoscopes.

A detailed illustration of the crane lifting system with compound cables is provided in Figure 5.8. In this illustration, the MOD student is examining all working members of the hoist system, including the attachment surfaces, the cabling systems, and the pinion wheels that govern the cables during the lifting process.



Figure 5.8: Crane Examination Techniques – Cable Lifting Systems. In this illustration the student is working through a prescribed series of surface tests that are required to characterize the functional operating capacity of the crane. These tests are used to develop a working model of how the hoist should be used given the approximate working capacity of the system and the remaining service life of duty cycle for the crane. The student is also shown methods for lubricating the cable system to minimize surface wear and cable friction.

Examination of the under-carriage for the crane system is illustrated in Figure 5.9a,b. Within this sequence, each student follows a sequence of testing operations beginning with basic capillary methods and magnetic identification methods. In this example, a portion of the crane has been altered and re-welded to correct a structural problem. The section is identified with red paint (surface primer). In addition, the structure includes new surface welds that require certification from the mobile team members. The students work with a fabrication specialist in Figure 5.9b to discuss the welding methods used to rebuild the undercarriage and further strengthen the frame characteristics of the hoisting system. The specialist describes the fabrication methods used and discusses the structural preparation that was required prior to the formal welding sequence.

The examination of the cabling system for strength and strand deformation is shown in Figure 5.10a-b for cable attached to the main pinion systems and in Figure 5.10c-d for cables that control the vertical



Figure 5.9a: Crane Examination Techniques – Undercarriage Examination.



Figure 5.9b: Crane Examination Techniques – Fabrication Methods. The fabrication specialist is shown (right) working with the MOD student (left) to identify the methods used to repair the structure.



Figure 5.10a: Crane Examination Techniques – Pivot Assemblies and Pinion Systems.



Figure 5.10b: Crane Examination Techniques – Cables and Turning Radius.



Figure 5.10c: Crane Examination Techniques – Cable Surface Conditions.



Figure 5.10d: Crane Examination Techniques – Horizontal Examination.

control systems. The figure shows the MOD students working through a series of physical tests that are required for Level I characterization. As indicated in Figure 5.11, the crane includes long sections of cable that require detailed magnetic examination for cable fatigue and probable service life determination. The cable lengths are tested with the Intros systems and with semi-circular transducer systems connected to the Sonatest and StressTel ultrasonic defectoscopes. The examination is required to check for elongation of the metal strands that indicates unusual wear or stretching in the main cable system.



Figure 5.11: Crane Examination Techniques – Load Testing Sequence. In this illustration, the MOD students are preparing the crane for a sequential load test that begins with a static load and ends with a pulsed load of a known weight. The static test is usually designed for 1.8 to 2.2 times above the rated service capacity of the crane. When this test is performed, the cable and hoisting mechanism is gauged to determine the stress induced during the lifting sequence. The pulsed test includes a series of lifts and drops that simulate a jerking process. The pulse generates sharp tensions on the crane cable that may be analyzed for elongation and probable deformation or loss of cross-section.

The examination of the surface leveling functions and lifting pivot systems is shown in Figure 5.12a,b. In this illustration, the MOD students are shown how the crane is stabilized during the lifting sequence using the jack assemblies that tie the understructure of the crane to the ground. In Figure 5.12b, the students examine the main hoisting assembly and are shown the safety features that are

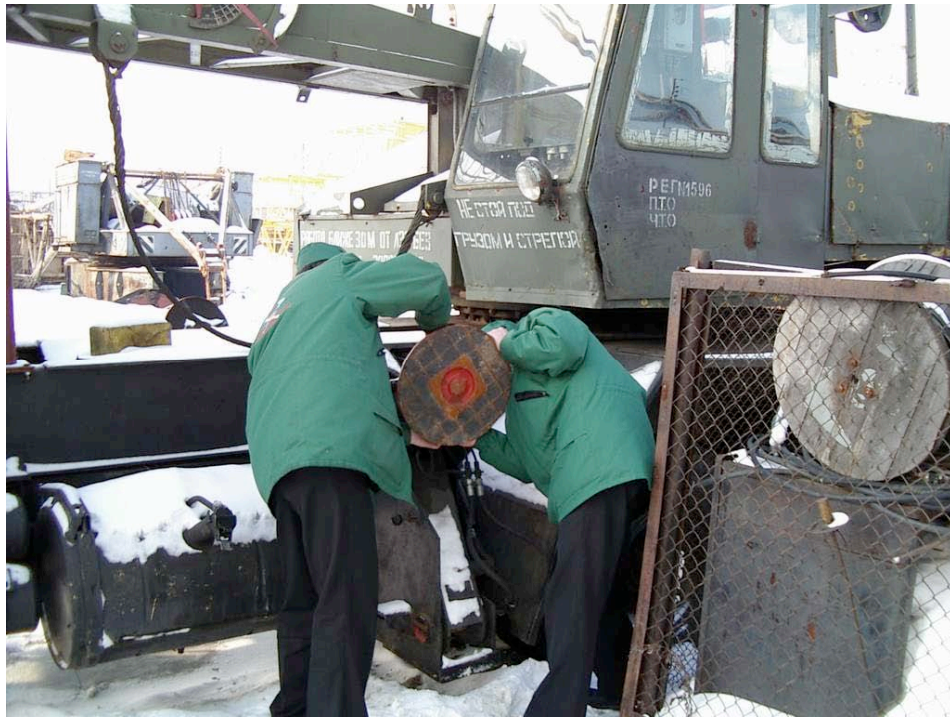


Figure 5.12a: Crane Examination Techniques – Extending the Leveling Foot.



Figure 5.12b: Crane Examination Techniques – Examining the Lifting Pivot.

used to ensure that the lift is stable and properly secured to the turnbuckle assembly. The crane examination includes pressure testing for the main hydraulic lines and bearing surfaces. In Figure 5.13, the students are examining the bearing pivot for the surface leveling system on the main crane assembly. The inspection also requires each student to understand how the hydraulic systems are used to move the jack assemblies into place and the suitable methods for testing the bursting capacity of the hydraulic lines (rubber and metal line assemblies).



Figure 5.13: Crane Examination Techniques – Hydraulic Lines and Bearing Surfaces.

The examination of the hydraulic cylinders that govern the movement of the crane boom is shown in Figure 5.14a,b. In this illustration, the students examine the bearing surfaces and pivot assemblies adjacent to each cylinder assembly. The examination is required to identify bearing pins that require replacement and lubrication requirements for the extended service life of the hoisting system.



Figure 5.14a: Crane Examination Techniques – Hydraulic Cylinders.



Figure 5.14b: Crane Examination Techniques – Hydraulic Friction Points.

6.0 Conclusions

The CTD-LIMS system is designed to provide a detailed framework for the storage, archive, and digital transfer of binary data from source instruments in the MOD fixed and mobile laboratory system. The software utilizes open-source procedures as a technique to minimize the service and maintenance costs, and maximize compatibility across the various platforms that are used by the 12th Main Directorate. As described in this technical documentation, the LIMS is operational and is currently being used by MOD in their fixed laboratory operations for certification and attestation. This includes the testing of specimen samples using scanning electron optics, x-ray fluorescence spectroscopy, and optimal investigation of metallographic surfaces using micro-hardness methods for non-destructive testing. The LIMS is also used for the application of linear, cyclic, and surface tensile/compressive testing for destructive modeling. In the development process, the LIMS works with the local instrumentation to exchange and archive information from the local host system. For the scanning electron microscope, the LIMS works with the Philips SQL server and receives information in an Oracle/Sybase format for secure database management. The database includes images from the formal electron optical scanner as well as spectrographic data and graphical images or tables from the FEI scanner that is integrated within this system. The combined data records are then transmitted to the LIMS, and parsed according to the application and technical requirements for the testing sequence. When formal tests for Gosgorteknadzor are required, all records from the electron optical scanner are recorded (including the primary element records that are required to support the attestation report). The LIMS database includes the high and low-resolution images, as well as, the spectrographic models that have been considered for the testing sequence. For the x-ray fluorescence investigation, the database includes all tests that have been applied to the sample and the related standards that have been used to calibrate the system. The LIMS records include the metadata pre-processing standards that have been employed for the investigation, and the Struers sample preparation data that documents the chain-of-custody for the sample. The chain-of-custody information is stored by specimen identification code and by testing reference. The references include all information concerning the analysts that have examined the sample, and the related pre-investigation procedures that have been applied to the specimen. In effect, this maintains a complete data record from the time the sample was acquired during the field investigation (the destructive removal of the surface from the crane, hoist, or high-pressure system) to the period when the sample was logged into the system by the investigation team. The x-ray fluorescence reports include detailed spectrographic models for primary element identification -- including detailed data for the distribution and frequency of the element that is under investigation. For the investigation of metallographic thin-section samples, the LIMS retains all information that may be attributed to the processing and mounting of the sample. This includes the destructive removal of sample materials from the crane, hoist, or high-pressure system, as well as, the polishing techniques that have been applied to the specimen to amplify the optical investigation. The polished surface is then mapped by position (within sample location) and registered within the LIMS. The reports include high-magnification image processing with multispectral classification for the surface feature. The classification and the related images are stored within the CTD-LIMS and appear within the final RSS and Gosgorteknadzor certification reports. Reports of this format were presented to the US Delegation to St. Petersburg in February 2004. For the destructive testing sequence, the LIMS retains all linear and cyclic information that is required to document the tensile and compressive features for the sample. This includes the detailed sample preparation procedures that are required to construct the convolution model. During the sample preparation sequence, MOD uses low temperature methods to remove material from the field specimen. This procedure is required to minimize the effects from temperature-induced bias during the tension-compression testing sequence. At the present time, ALL is working with MOD to support specific testing procedures, instruments, and

removal standards that are required to prepare the samples with minimum bias. These methods will be presented to DTRA for their technical review.

The LIMS uses independent workstations to pre-process and organize data according to RSS and Gosgorteknadzor standards. Within the fixed laboratory system, any workstation may be used that has an installed StarLIMS client. At the present time, MOD utilizes six floating licenses for this effort. The licenses are dynamic and may be used within any of the testing laboratories with no loss of function or related technical information. In addition, the LIMS supports mobile docking stations that provide the ideal framework for transferring information from field instruments that support data-logging capabilities. The mobile docking approach yields numerous benefits for MOD: first, the technique is highly generalized. Hence, any instrument that can transmit data onto a laptop can also transfer information into the LIMS using one or more of the SQL or PL/SQL procedures described in Section 2 and Section 3 of this manuscript. Second, the docking configuration allows the user to operate LIMS procedures in a purely mobile environment – without the technical requirements or administration guidelines required for the fixed laboratory. Hence, the two laboratories (fixed and mobile) can operate as independent facilities. This is required for specific investigations that are either independent or delineated with respect to mobile investigations or MOD procedures. Third, the algorithms are shown to be web-client based for open source transfer of digital information using *html* and other standard protocols such as TCP/IP and the equivalent SQL/*SMTP*.

The LIMS examples provided in Section 2 and Section 3 should be used in conjunction with the SQL productivity tools shown in Appendix A-G. Although the LIMS graphical user interface is strictly controlled with respect to MOD security and processing requirements, the PL/SQL and SQL algorithms are completely open for design and modification. Indeed, the data transfer utilities used within the main LIMS server are written in SQL and PL/SQL to access the data from the various instruments and servers on the CTD network. In this design, the CTD-LIMS uses primary SQL, whereas, the Oracle specific CTD materials database uses PL/SQL to maintain the Oracle specific features requested by MOD. The algorithms are shown to be open source and may be used to minimize future requirements for software upgrades and proprietary software. All code examples use ISO compliant SQL and PL/SQL to maximize transfer of information throughout the fixed and mobile laboratory system.

As presented in this technical manuscript, the MOD certification and training programs are on-schedule for final examinations from Gosgorteknadzor. The students are completing their Level II attestations and will continue their training program as required for the safe and secure operation of the MOD facility. We will work closely with DTRA to assist their technical staff in the future support of this training and certification program.

7.0 References

Acoustic Emissions. Terms, Definitions and Acronyms. (GOST 27655-88). General Reference: Assessment of Mechanical Attributes of Materials Based on the Acoustic Emission Method. (RD-50-568-85).

ASTM, Guide E2066-00 Standard Guide for Validation of Laboratory Information Management Systems, 2001.

ASTM Standard E-622 Guide for Developing Computerized Systems.

ASTM Standard E-623 Guide for Developing Functional Requirements for Computerized Systems.

ASTM LIMS Guide. In 1994 *Book of ASTM Standards; American Society for Testing and Materials; 1994*; Vol. 14.01.

Bangia, Ramesh, *Database Management Systems (DBMS)* (2000), A. H. Wheeler Publishing Co Ltd.

Batory D. and Thomas J., "P2: A Lightweight DBMS Generator", *Journal of Intelligent Information Systems*, 9, 107-123 (1997).

Beresniewicz, John, Feuerstein, Steven, Dye, Charler, *Oracle Built in Packages* (1998) Cambridge: O'Reilly & Associates, Inc.

Bonner A., Shrufi A. , and Rozen S. *LabFlow-1: A database benchmark for high-throughput workflow management*. In *Proceedings of the 5th Int. Conference on Extending Database Technology (EDBT96)*, Avignon, France, 3 1996.

Brown, Philip J. *Measurement, Regression, and Calibration* (1994), Oxford: Clarendon Press.

Cascio, Joseph, Woodside, Gayle and Mitchell, Philip, *ISO14000 Guide: The New Environmental Management Standards* (1996), Milwaukee: American Society for Quality Control (ASQC).

Comprehensive Inspection of Crane Tracks for Lifting Machinery. Section 1. RD-10-138.

Decision Regarding the Individual Functions of the State Mining and Industrial Inspectorate of Russia and the Inspection Office of the State Technical Inspectorate of the Armed Forces of the RF. (MOD order 39, 1994).

Dietrich, C.F. *Uncertainty, Calibration and Probability: The Statistics of Scientific and Industrial Measurement Second Edition* (1997), Bristol: Adam Hilger.

Elevator Installation and Operational Safety Rules. (Order_1 of Gostekhnadzor RF, 11 February 1992).

EPA Good Automated Laboratory Practices, Recommendations for Ensuring Data Integrity in Automated Laboratory Operations. Washington, D.C. August, 1995.

FDA. Quality System Regulation. 21 Code of Federal Regulations part 820, 1996

Feigenbaum, Armand V. *Total Quality Control, Third Edition* (1983), New York: McGraw Hill.

Feuerstein, Steven, *Oracle PL/SQL Programming, Third Edition* (2002), Cambridge: O'Reilly & Associates, Inc.

Feuerstein, Steven, *Advanced Oracle PL/SQL Programming with Packages* (1996) Cambridge: O'Reilly & Associates, Inc.

Feuerstein, Steven, *Oracle PL/SQL Best Practices* (2001) Cambridge: O'Reilly & Associates, Inc.

Flanagan, David, *JavaScript The Definitive Guide Third Edition* (1998), Cambridge: O'Reilly & Associates, Inc.

Garfield, Frederick M. *Quality Assurance Principles for Analytical Laboratories Second Edition* (1991) Baltimore: AOAC.

Geary, David M. *Graphic JAVA, Mastering the JFC, Volume II Swing* (1999) Palo Alto: Sun Microsystems, Inc.

Gennick, Jonathan, *Oracle SQL *Plus: The Definitive Guide* (1999), Cambridge: O'Reilly & Associates, Inc.

Goodman, Danny, *Dynamic HTML: The Definitive Reference (2nd Edition)* (2002), Cambridge: O'Reilly & Associates, Inc.

Goodman N., Rozen S., and Stein L. Labbase: A database to manage laboratory data in a large-scale genome-mapping project. *IEEE Computers in Medicine and Biology*, 14:702 -- 709, December 1995.

GOS Standard Technical Specifications for the Repair of Industrial Steam and Hot Water Boilers (Order of Gosgortekhnadzor RF, 4 July 1994). General Reference: Methodological Procedures for Technical Diagnostics of Boilers with a Working Pressure Rate of ≤ 4.0 MPa. RD 34.17.435-95.

Greenwald, Rick and Milbery, James, *Oracle 9ias Portal Bible* (2001), New York: Hungry Minds, Inc.

Guidance for Federal State Mining and Industrial Inspectorate Regulations and Technical Documents Related to the Development, Production, Operation, Upgrading and Renovation of Equipment and Systems under the Oversight of the State Technical Inspectorate of the Armed Forces of the RF (RTB-95). General Reference: (Order 214 of the RF MOD, 1995).

Hansen Gary W. and Hansen, James V. *Database Management and Design, Second Edition* (1995), New York: Prentice Hall.

Harold, Elliotte Rusty, *JAVA I/O* (1999), Cambridge: O'Reilly & Associates, Inc.

Harrington, H. James *Total Improvement Management* (1994) Milwaukee: American Society for Quality Control (ASQC).

Hilton, Mary D (1994). *Laboratory Information Management Systems: Development and Implementation for a Quality Assurance Laboratory* New York: Marcel Dekker.

Hoffer, Jeffrey A, Prescott, Mary B., McFadden, Fred R. *Modern Database Management (6th Edition)* (2002) New York: Prentice Hall.

Huber, L. *Validation of Automated Computer Systems*; InterPham Press: Buffalo Grove, IL, 1995; pp 4-58

Huber, Ludwig, *Validation of Computerized Analytical Systems* (1995) Boca Raton: CRC Press LLC.

Interim Program for Technical Diagnostics of DKVR and DG Steam Boilers Based on the Acoustic Emission Method. (Central Design and Research Institute of Boilers and Turbines, 19 June 1996).

Kyte, Thomas, *Effective Oracle by Design* (2003), New York: McGraw-Hill Osborne Media.

Kyte, Thomas, *Expert One-on-One Oracle* (2001), New York: Wrox Press Ltd.

Land, Russ, Fire in the Hole! Measuring the Value of Continuous Improvement, QP, Jan 2001 pg 89-93.

Lifting Crane Installation and Operational Safety Rules. (Order _12 of Gostekhnadzor RF, 12 May 1993).

Lifting Machinery. Metallic Structures. Radiation Monitoring. General Provisions. RD RosEK-002-96. General Reference: Ultrasonic Monitoring RD RosEK-001-96.

Litwin, Witold, Morzy, Vossen, Gottfried, *Advances in Databases and Information Systems* (1998), Berlin: Springer-Verlag.

Loney, Kevin and Koch, George, *Oracle 9i: The Complete Reference* (2002) New York: McGraw-Hill Osborne Media.

Lyon, Douglas A. *Image Processing in JAVA* (1999), Upper Saddle River: Prentice Hall.

Mahaffey, Richard R. & Reinhold Van Nostrand, *LIMS: Applied Information Technology for the Laboratory* (1990), Dordrecht: Kluwer Academic Publishers.

Major Safety Requirements for Electric Bridge and Gantry Cranes Load Limits. RD-10-118.

McDowall, Robert D., *Laboratory Information Management Systems, Concepts, Integration, Implementation* (1987), New York: John Wiley & Sons.

McDowell RD. Operational Measures to Ensure Continued Validation of Computerized Systems in Regulated or Accredited Laboratories, Lab Automat Info Manage 31, 1995.

McFarlane, Nigel, *Instant JavaScript* (1997), New York: Wrox Press Ltd.

McLaughlin, Gregory C. *Total Quality in Research and Development* (1995) Boca Raton: St Lucie Press.

Methodological Procedures for Determining the Remaining Service Life of Potential Hazardous Sites Subject to Gostekhnadzor Oversight. (Order _ 57 of Gostekhnadzor RF, 17 November 1995). General Reference: National Research Institute of Lifting and Transportation Machinery (12 July 1991).

Methodological Procedures for Inspecting Lifting Machinery Whose Service Life Has Expired. Section 5. Bridge and Gantry Cranes. (RD-10-1112-5-97). General Reference: All-purpose, Self-propelled Derrick Cranes. (RD-10-112-2-97). Separate Provisions: (RD-10-112-96).

Methodological Procedures for Technical Diagnostics and Extending the Service Life of Pressurized Vessels. (RD 34.17.439-96).

Methodological Procedures. Ultrasonic Monitoring of Metal Structure Welds When Inspecting Lifting Cranes. (RD IKTs KRAN-001-92).

Methodological Recommendations. The Use of the Acoustic Emission Method to Establish Destructive Strength Attributes. (MR 204-87).

Mishra, Sanjay and Beaulieu, Alan, *Mastering Oracle SQL* (2002), Cambridge: O'Reilly & Associates, Inc.

Morris, Alan S. *Measurement and Calibration Requirements for Quality Assurance to ISO 9000* (1998), New York: John Wiley & Sons.

Nakagawa, Allen S. *LIMS Implementation and Management*. Royal Society of Chemistry, Cambridge, 1994.

National Institute for Science and Technology. *Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards*, 1996.

Nilsen, Clifford L. *Managing the Analytical Laboratory Plain and Simple* Boca Raton: CRC/ Interpharm Press.

Non-destructive Examination. Ultrasonic Transducers. Measurement Methods for Major Parameters. (GOST 23702-85). General Reference: (GOST 17410-77).

Non-destructive Examination. Seamless, Cylindrical Metal Pipes. Ultrasonic Defectoscopy Methods. (GOST 17410-78).

Non-destructive Examination. Ultrasonic Defectoscopy. Measurement Methods for Major Parameters. (GOST 23667-85).

Non-destructive Examination. Welds. Ultrasonic Methods. (GOST 14782-86).

Oaks, Scott & Wong, Henry, *JAVA Threads Second Edition* (1999), Cambridge: O'Reilly & Associates, Inc.

Odewahn, Andrew, *Oracle Web Applications* (1999), Cambridge: O'Reilly & Associates, Inc.

Oelker, Greg, *How Doves LIMS Use Help Laboratory Function*. SC&A.

Ozsu, M. Tammer and Valduriez, Patrick, *Principles of Distributed Database Systems, Second Edition*(1999) New York: Prentice Hall.

Passenger and Freight Elevators. Methodological Procedures for Inspecting the Status of Elevators Whose Standard Service Life Has Expired. (Order of Gosgortekhnadzor RF, 22 July 1994).

Paszko, Christine, Miller, Tom, Vranken, Russ, *Plugging Into LIMS: Evolution and Advances in the Age of PCs How Personal Computers Have Revolutionized Laboratory Automation and Data Management*
<http://www.atlab.com/Events%20and%20Publications/Publications/pluginintolims.html>.

Paszko, Christine, Turner, Elizabeth and Hilton, Mary D (2001). *Laboratory Information Management Systems Revised & Expanded Second Edition*. New York: Marcel Dekker.

Paszko, Christine Extending LIMS Functions Over The Internet, Accelerated Technology Laboratories, Inc. AOAC International, April 1998.

Paszko, Christine and Pugsley, Carol, Considerations in Selecting a Laboratory Information Management System (LIMS), American Laboratory, September, Volume 32 No 18, 2000.

Powell, Thomas & Schneider, Fritz, *JavaScript: The Complete Reference* (2001) New York: McGraw-Hill Osborne Media.

Pressurized Vessel Installation and Operational Safety Rules. (Order _ 20 of Gostekhnadzor RF, 18 April 1995). General Reference: Steam and Hot Water Pipes. (RD-03-29-93).

Procedure for Issuing Special Permits to Conduct High Hazard Operations at State Technical Inspectorate Sites of the Armed Forces of the Russian Federation. (Decision 152/190 of the State Technical Inspectorate of the Armed Forces of the RF, 31 March 1995.

Ramakrishnan, Raghu, Gehrke, Johannes, *Database Management Systems, Third Edition* (2003), New York: McGraw Hill – Higher Education.

RD RosEK-01-013-97. Regulation on the Procedure for Specialist Training and Certification for Expert Inspections and Technical Diagnostics.

Regulation on the Technical Diagnostics System for Industrial Steam and Hot Water Heaters. (Central Design and Research Institute of Boilers and Turbines, 27 March 1992; National Research Institute of Nuclear Material, 20 March 1992; Central Research Institute of Machine Building Technology, 19 March 1992).

Requirements for Non-destructive Examination and Diagnostic Laboratories (RD-RosEK-005-96). General Reference: Visual and Measurement Inspection Procedures (RD-34.10.130-96).

Rob, Peter, Coronel, Carlos, *Database Systems: Design, Implementation, and Management, Fifth Edition* (2001), Boston: Course Technology.

Rozenshtein, David, Abramovich, Anatoly, Birger, Eugene, *Optimizing Transact-SQL: Advanced Programming Techniques* (1997) New York: SQL Forum Press.

Rules for Organizing and Conducting Acoustic Emission Monitoring of Vessels, Equipment, Boilers and Process Pipes. (RD-03-131-97. Order _ 44 of Gostekhnadzor RF, 11 November 1996).

Saltor, F. Ramos, I and Alonso G., editors *The Proceedings of EDBT98, the 6th International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 193-197. Springer, 1998.

Silberschatz, Abraham, Korth, Henry F., Sudarshan, S, *Database System Concepts, Fourth Edition* (2002), New York: McGraw Hill – Higher Education.

Stafford, J.E.H, *Advanced LIMS Technology, Case Studies and Business Opportunities* (1995), Dordrecht: Kluwer Academic Publishers.

Steam and Hot Water Boiler Installation and Operational Safety Rules. (Order _ 12 of Gostekhnadzor RF, 12 May 1993). Also, Durability Calculations and Testing. The Use of Acoustic Emission Method to Monitor Pressurized Vessels and Pipes. (MR 204-86).

Teague, Jason Cranford, *DHTML For the World Wide Web* (1998), Berkley, CA: Peachpit Press.

Turner, Elizabeth, Paszko, Christine and Kolva, Don, Implementing a Laboratory Information Management System (LIMS) in an Army Corps of Engineer's Water Quality Testing Laboratory, *Journal of the Association of Laboratory Automation* Vol 6 No. 5 November 2001 pages 60-63.

Urman, Scott, *Oracle 9i PL/SQL Programming* (2001), New York: McGraw-Hill Osborne Media.

Vesterli, Sten E. *Oracle Web Applications 101* (2001) New York: McGraw-Hill Osborne Media.

Weinberg, Spelton & Sax, *GALP Regulatory Handbook* (1994), Lewis Publishers, CRC Press.

Woodall, Jack, Rebuck, Deborah K. and Voehl, Frank, *Total Quality in Informational Systems and Technology* (1996), Boca Raton: St Lucie Press.

Laboratory Instruments and Data Management Systems: Design of Software User Interfaces and End-User Software Systems Validation, Operation, and Monitoring; Approved Guideline; NCCLS: Wayne, PA, 1995; GP-19A.

Appendix A: LIMS Time-Date Formats

The LIMS uses specialized functions for formatting the technical reports that are required by RSS and Gosgorteknadzor. Within this technical appendix, the algorithms required for managing dates and times are provided. These methods are used for the “time-stamp” conventions that have been displayed throughout this report, and are used within the StarLIMS and the CTD materials database. The time indexes are placed within the metadata records for the 12th Main Directorate. As a result, these algorithms are required for the final technical reports that include the QA/QC models and the chain-of-custody references.

A.1 Convert a String To a Date – Day First

Each date contains the: century, year, month, day, and hour, minute and second. Since date formats can fluctuate, it is necessary to allow for varying input parameters. This example accepts, as a parameter, any string in a day-month-year format and returns a final date that is organized according to MOD requirements. The following section demonstrates a variety of functions and procedures that can be used to convert to and from the date format.

Create or replace Function StringToDate(pString varchar2) return date is

```

returnval date;

BEGIN
    if pstring is not null then
        returnval:=to_date(pString);
    end if;
    return (returnval);

exception
    when others then
    begin
        returnval:=to_date(pString,'dd/MM/yy');
        return (returnval);

    exception
        when others then
        begin
            returnval:=to_date(pString,'dd/MM/yyyy');
            return (returnval);

        exception
            when others then
            begin
                returnval:=to_date(pstring,'dd-MON-yyyy');
                return (returnval);
            exception
                when others then
                begin
                    returnval:=to_date(pstring,'dd-MONTH-yyyy');
                    return (returnval);
                exception
                    when others then
                    begin
                        returnval:=to_date(pstring,'ddMONTHyyyy');
                        return (returnval);

                    exception
                        when others then
                        begin
                            returnval:=to_date(pstring,'ddMONyy');
                            return (returnval);
                        exception
                            when others then

```



```

        begin
        returnval:=to_date(pstring,'ddMONyyyy');

        return (returnval);
    exception
        When others then
            return(NULL);
    end; end; end;
end; end; end;
end;

end StringToDate;
/

```

The following SQL statement may be used to validate the procedure. The result is a standard string date in the CTD format:

```

SQL> select stringtodate('01-01-2004') from dual;

STRINGTODATE
-----
01-JAN-04

```

A.2 Convert a String To a Date – Month First

```

Create or Replace Function StringToDateMonthFirst(pString varchar2)
return date is returnval date;

BEGIN
    if pstring is not null then
        returnval:=to_date(pString,'MM-dd-yy');
    end if;
    return (returnval);
exception
    when others then
        begin
            returnval:=to_date(pString,'month-dd-yyyy');
            return (returnval);
        exception
            when others then
                begin
                    returnval:=to_date(pString,'MM-dd-yyyy');
                    return (returnval);
                exception
                    when others then
                        begin
                            returnval:=to_date(pstring,'MM/dd/yyyy');
                            return (returnval);
                        exception
                            when others then
                                begin
                                    returnval:=to_date(pstring,'Mon/dd/yyyy');
                                    return (returnval);
                                exception
                                    When others then
                                        Return(StringToDateMonthFirst(pString));
                                        null;
                                    end;
                                    end;
                                    end;
                                    end;
                                end StringToDateMonthFirst;
                                /

```

The following SQL statement may be used to validate the procedure. The result is a standard string date in the RSS format. This format uses the day-month-year convention:

```
SQL> select stringtodatemonthfirst('Jan-28-2004') from dual;

STRINGTODATEMONTHFIRST
-----
28-JAN-04
```

A.3 Convert a String To a Date – Year First

```
Create or Replace Function StringToDateYearFirst(pString varchar2) return date is
    returnval date;
begin
    if pstring is not null then
        returnval:=to_date(pString,'yy-MM-dd');
    end if;
    return (returnval);
exception
    when others then
        begin
            returnval:=to_date(pString,'yyyy-MM-dd');
            return (returnval);
        exception
            when others then
                begin
                    returnval:=to_date(pString,'yyyy/MONTH/dd');
                    return (returnval);
                exception
                    when others then
                        begin
                            returnval:=to_date(pstring,'yyyy-Mon-dd');
                            return (returnval);
                        exception
                            when others then
                                begin
                                    returnval:=to_date(pstring,'yy-Mon-dd');
                                    return (returnval);
                                exception
                                    When others then
                                        Return(StringToDateMonthFirst(pString));
                                end; end;
                            end; end;
                        end StringToDateYearFirst;
                    /
```

The following SQL statement may be used to validate the procedure. The result is a standard string date in the RSS format. The year 2004 has been relocated to the proper position.

```
SQL> select stringToDateYearFirst('2004-Jan-28') from dual;

STRINGTODATEYEARFIRST
-----
28-JAN-04
```

A.4 Convert String To Date time

```
Create or Replace Function StringToDateTime(pString varchar2) return date is
    returnval date;
begin
    if pstring is not null then
        returnval:=to_date(pString);
    end if;
    return (returnval);
exception
    when others then
```

```

begin
    returnval:=to_date(pString,'dd/MM/yy hh24:mi');
    return (returnval);
exception
    when others then
    begin
        returnval:=to_date(pString,'dd/MM/yyyy hh24:mi');
        return (returnval);
    exception
        when others then
        begin
            returnval:=to_date(pstring,'dd-MON-yyyy hh24:mi');
            return (returnval);
        exception
            when others then
            begin
                returnval:=to_date(pstring,'dd-MONTH-yyyy hh24:mi');
                return (returnval);
            exception
                when others then
                begin
                    returnval:=to_date(pstring,'ddMONTHyyyy hh24:mi');
                    return (returnval);
                exception
                    when others then
                    begin
                        returnval:=to_date(pstring,'ddMONyy hh24:mi');
                        return (returnval);
                    exception
                        when others then
                        begin
                            returnval:=to_date(pstring,'ddMONyyyy hh24:mi');
                            return (returnval);
                        exception
                            When others then
                                Return(StringToDateYearFirst(pString));
                    end; end; end;
                end; end; end;
            end;
        end StringToDateTime;
    /

```

The following SQL statement may be used to validate the procedure. The result is a standard string date in the Gosgorteknadzor format. The string includes the additional time data for QA/QC modeling. This format is used within the StarLIMS and the CTD materials database.

```

SQL> select stringtodatetime('30/04/200410:45') from dual;

STRINGTODATETIME('30/04/200410:45')
-----
4/30/2004 10:45:00 AM

```

A.5 Convert Date to a String

It is frequently necessary to convert a date to a string. This is used when displaying a specific format. The following example demonstrates a method to convert a date to a string:

```

Create or Replace Function DateToString(pDate date) return varchar2 is
begin
    return(to_char(pDate,'dd-Mon-yyyy'));
end DateToString;

```

```

SQL> select datetostring('28-Jan-2004') from dual;

DATETOSTRING('28-JAN-2004')
-----
28-Jan-2004

```

A.6 Convert Date Time to a String

Converting a date time is frequently required for RSS technical reports. This function may be used to perform this basic operation:

```
Create or Replace Function DateTimeToString(pDate date) return varchar2 is
    BEGIN
        return(to_char(pDate,'dd-Mon-yyyy hh24:mi'));
    end DateTimeToString;
```

```
SQL> select datetimetostring(sysdate) from dual;
```

```
DATETIMETOSTRING(SYSDATE)
```

```
-----
24-Feb-2004 09:36
```

A.7 Convert a date/time to GMT

The GMT time is required for all data that is forwarded to the International Organization of Standardization (ISO). In general, this time form is not required by MOD, but is used by RSS and Gosgorteknadzor for forwarding information to the European certification bureaus.

```
Create or Replace Function ConvertTimeToGMT(pTime varchar2,pOffset varchar2 default 'EST')
return date IS
```

```
    RETURNVAL DATE;
    BEGIN
```

```
    RETURNVAL:=to_date(pTime,'dd-mon-yyyy:hh24mi');
    IF pOffset = 'EST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'EST','GMT') into
        RETURNVAL FROM dual;
```

```
    ELSIF pOffset = 'CST' THEN
        SELECT NEW_TIME(RETURNVAL,'CST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'MST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'MST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'PST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'PST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'AST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'AST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'BST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'BST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'HST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'HST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'NST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'NST','GMT') into RETURNVAL FROM dual;
    ELSIF pOffset = 'YST' THEN
```

```
        SELECT NEW_TIME(RETURNVAL,'YST','GMT') into RETURNVAL FROM dual;
    END IF;
    return(returnval);
```

```
exception when OTHERS then
    begin
```

```

RETURNVAL:=to_date(pTime,'dd-mon-yyyy:hh24:mi');
IF pOffset = 'EST' THEN
    SELECT NEW_TIME(RETURNVAL,'EST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'CST' THEN
    SELECT NEW_TIME(RETURNVAL,'CST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'MST' THEN
    SELECT NEW_TIME(RETURNVAL,'MST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'PST' THEN
    SELECT NEW_TIME(RETURNVAL,'PST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'AST' THEN
    SELECT NEW_TIME(RETURNVAL,'AST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'BST' THEN
    SELECT NEW_TIME(RETURNVAL,'BST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'HST' THEN
    SELECT NEW_TIME(RETURNVAL,'HST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'NST' THEN
    SELECT NEW_TIME(RETURNVAL,'NST','GMT') into RETURNVAL FROM dual;
ELSIF pOffset = 'YST' THEN
    SELECT NEW_TIME(RETURNVAL,'YST','GMT') into RETURNVAL FROM dual;
END IF;
return(returnval);
end;
end ConvertTimeToGMT;

```

The following SQL statement may be used to validate the procedure. The result is a standard string date in the ISO format – indexed to GMT time.

```
SQL> Select ConvertTimeToGMT('01-Jan-2004:11:00') from dual
```

```
CONVERTTIMETOGMT('01-JAN-2001:11:00','EST')
```

```
-----
1/1/2004 4:00:00 PM.
```

A.8 Convert Zulu To Date

The following functions are used to manage zulu time conventions used in military service. The models utilize the ISO standards for time-date exchange:

```

Create or Replace Function ConvertZuluToDate(pZulu varchar2) return date
IS
    returnval date:=SYSDATE;
BEGIN
    returnval:=to_date(pZulu,'ddhh24mi"z"MonYYYY');
    return (returnval);
exception
    when value_error then
        return (sysdate);
end ConvertZuluToDate

```

The following SQL statement may be used to validate the procedure. The result is a standard string parsed from the original zulu representation.

```
SQL> Select ConvertZulutoDate('011101zJan2004') from dual
```

```
CONVERTZULUTODATE('011101ZJAN2004')
```

```
-----  
1/1/2004 11:01:00 AM
```

The reverse operation (moving from date back to zulu) is shown as:

```
Create or Replace Function ConvertDateToZulu(pDate date) return varchar2 is
```

```
begin  
return (to_char(pDate,'ddhh24mi"z"MonYYYY'));  
end ConvertDateToZulu;
```

The following SQL statement may be used to validate the converse procedure. The result is the standard zulu date representation:

```
SQL> select convertdatetozulu('01-MAR-2004') from dual;
```

```
CONVERTDATETOZULU('01-MAR-2004')
```

```
-----  
010000zMar2004
```

A.9 Convert Custom Date

Frequently, dates do not follow a pre-defined format. For example, time/date information that is forwarded to the LIMS from a specific vendor. When this occurs, it is necessary to convert the sting into the standard CTD format:

```
Create or Replace FUNCTION convertCustomDate(pDateString Varchar2) return Date  
IS  
BEGIN
```

```
Return to_date(pDateString,'DD-MM-YYYYHH:MI AM "ET"');  
End convertCustomDate;
```

The following SQL statement may be used to validate the custom conversion procedure. The result is the standard CTD date representation:

```
SQL> select convertcustomdate('01-01-2004 10:00 AM ET') from dual;
```

```
CONVERTCUSTOMDATE('01-01-200410:00AMET')
```

```
-----  
01-JAN-04 10:00:00 AM
```

Appendix B: Arrays and Temporary Data Storage

The LIMS uses specialized functions for holding critical information. The functions use array data structures to manage the digital information required for the specific processing task. In general, any process that sorts information, or parses data, utilizes an array structure to manage intermediate processing tasks. In this appendix, we examine the main utility functions that are required to define associative arrays, user defined arrays, and custom configurations for time and date management.

B.1 Associative Arrays

The associative array is a predefined map or table. Conceptually, an associative array is composed of a collection of keys, and a collection of values. In this organization, each key is associated with one value. The operation of finding the value (associated with a key) is called a lookup or indexing. The following example demonstrates the basic functionality of an associative array.

```
DECLARE
  TYPE machine_type IS TABLE OF VARCHAR2(500) INDEX BY VARCHAR2(64);
  machine_active machine_type;
  machine_obsolete machine_type;

  active NUMBER;
  obsolete NUMBER;
  ctr VARCHAR2(64);
BEGIN
  machine_active('Machine1') := 'AB123';
  machine_active('Machine2') := 'AC456';
  machine_obsolete('Machine3') := 'RE975';
  machine_obsolete('Machine4') := 'ZA345';
  active := machine_active.COUNT;
  obsolete:=machine_obsolete.COUNT;

  DBMS_OUTPUT.PUT_LINE ('COUNT ACTIVE = ' || active);
  DBMS_OUTPUT.PUT_LINE ('COUNT OBSOLETE = ' || obsolete);
  ctr := machine_obsolete.FIRST;

  DBMS_OUTPUT.PUT_LINE ('Obsolete Machine ID: ' || machine_obsolete(ctr));
  ctr := machine_obsolete.LAST;
  DBMS_OUTPUT.PUT_LINE ('Obsolete Machine ID: ' || machine_obsolete(ctr));
  ctr := machine_active.FIRST;

  DBMS_OUTPUT.PUT_LINE ('Active Machine ID: ' || machine_active(ctr));
  ctr := machine_active.LAST;
  DBMS_OUTPUT.PUT_LINE ('Active Machine ID: ' || machine_active(ctr));
END;
```

During execution, the procedure yields the following output:

```
COUNT ACTIVE = 2
COUNT OBSOLETE = 2
Obsolete Machine ID: RE975
Obsolete Machine ID: ZA345
Active Machine ID: AB123
Active Machine ID: AC456
```

B.2 User-Defined Type

The user-defined data types are composed of the built-in structures or previously declared objects. The objects may include both attributes and methods. Incomplete object types (without either attributes or methods) can be declared to enable dependent objects. The user-defined objects can be further organized to define a column in a relational table. This sequence may be extended to define an

object table (or to define a relationship to another data type). The following example demonstrates the creation and implementation of a user-defined type:

```
CREATE TABLE APP_TBL
(
  MACHINE_ID          NUMBER
  MACHINE_NAME        VARCHAR2(500)
  ROCKWELL_HARDNESS   NUMBER
)
```

Next types are created to enable access to components of the table that was just created.

```
CREATE OR REPLACE type application_tbl
as table of application_type
```

```
CREATE OR REPLACE type application_type
as object
( MACHINE_ID NUMBER,
  MACHINE_NAME VARCHAR2(100),
  ROCKWELL_HARDNESS NUMBER)
```

```
CREATE OR REPLACE PACKAGE APPLICATION_SET AS
FUNCTION getApplication(MACHINEID IN NUMBER DEFAULT 0,
MACHINENAME IN VARCHAR2 DEFAULT NULL)
return application_TBL;
END APPLICATION_SET;
/
```

```
CREATE OR REPLACE PACKAGE BODY APPLICATION_SET AS
```

```
  FUNCTION getapplication (MACHINEID IN NUMBER DEFAULT 0,MACHINENAME IN VARCHAR2 DEFAULT
NULL) RETURN application_TBL
```

```
  IS
    l_data application_TBL;
```

```
  BEGIN
    select cast( multiset( select * from app_tbl WHERE MACHINE_ID=MACHINEID OR
MACHINE_NAME=MACHINENAME )
              AS application_TBL )
```

```
    into l_data
    from dual;
    return l_data;
```

```
  END;
END APPLICATION_SET;
/
```

```
DECLARE
DATA APPLICATION_TBL;
```

```
BEGIN
DATA:=APPLICATION_SET.GETAPPLICATION(1005,NULL);
FOR i IN 1..DATA.COUNT LOOP
```

```
    dbms_output.put_line(DATA(i).MACHINE_ID||' ' ||DATA(i).MACHINE_NAME||' '
' ||DATA(i).ROCKWELL_HARDNESS);
END LOOP;
END;
/
```

During execution, the procedure yields the following output:

```
1005 Machine A 5
```


Appendix C: Date Algebra

When dealing with temporal data, specific operations are required to edit the information. This includes indexing operations and arithmetic processes that are used to increment and update the process. The following utilities are required to perform simple operations on dates and times. These algorithms are automatically used within the LIMS during the chain-of-custody operations.

C.1 Add A Number Of Months From A Date

The following example adds the number of months specified in the parameter *months_shift* to the parameter passed to the function in *date_in*.

```
create or replace function new_add_months(date_in IN DATE,months_shift IN NUMBER)
Return Date
is
return_value DATE;
BEGIN
return_value:=Add_months(date_in,months_shift);
if date_in=Last_day(date_in)
then
return_value:=
least(return_value,to_date(to_char(return_value,'MMYYYY')||
to_char(date_in,'DD'),'MMYYYYDD'));
end if;
return return_value;
end new_add_months;
/
```

The following SQL statement may be used to validate the procedure:

```
SQL> select new_add_months('30-JAN-2004',1) AS NEWMONTH FROM DUAL;

NEWMONTH
-----
29-FEB-04
```

C.2 Last Day Of The Month

This function returns the last day of the month for the parameter *pDateString*.

```
Create or Replace FUNCTION lastDayOfMonth(pDateString Varchar2) return Date
IS
BEGIN
Return Last_day(pDateString);
End lastDayOfMonth;
```

The following SQL statement may be used to validate the procedure:

```
SQL> select lastDayOfMonth('30-JAN-2004') from dual;

LASTDAYOFMONTH('30-JAN-2004')
-----
31-JAN-04
```

C.3 Additions and the Last Day of the Month

Combining two built-in date functions, *Add_months* and *Last_day* are used in this process. The function returns a calculated value for the number of days (from the last day of the month). This is demonstrated in the following example:

```
Create or Replace FUNCTION lastDayOfMonth_ADDMONTHS(pDateString Varchar2 DEFAULT
SYSDATE,NUM_MONTHS NUMBER DEFAULT 1) return Date
IS
BEGIN
    Return Last_day(ADD_MONTHS(pDateString,NUM_MONTHS));
End lastDayOfMonth_ADDMONTHS;
/
```

The following SQL statement may be used to validate the procedure:

```
SQL> SELECT lastDayOfMonth_ADDMONTHS('01-JAN-2004',2) FROM DUAL;

LASTDAYOFMONTH_ADDMONTHS('28-JAN-2004',4)
-----
31-MAR-04
```

C.4 Days Until the End of the Month

This function returns the number of days until the end of the month from the parameter *pDateString*, which is specified when the function is initially applied.

```
Create or Replace FUNCTION daysLeftInMonth(pDateString Varchar2 DEFAULT SYSDATE) return
number
IS
BEGIN
    Return (Last_day(pDateString)-sysdate);
End daysLeftInMonth;
/
```

The following SQL statement may be used to validate the procedure:

```
SQL> select daysleftinmonth('04-Mar-2004') from dual;

DAYSLEFTINMONTH('04-MAR-2004')
-----
27.3448264
```

C.5 Next Day of the Month

This function returns the next day, specified by *day_name*, from *pDate*, which is passed into the function. This is required for leap-year calculations and month-by-month indexing.

```
Create or Replace FUNCTION theNextDay(pDate in date,day_name in varchar2) return date
IS
BEGIN
    Return next_day(pDate,day_name);
End theNextDay;
/
```

The following SQL statement may be used to validate the procedure:

```
SQL> select theNextDay('28-Feb-2004','Tuesday') from dual;

THENEXTDAY
-----
02-MAR-04
```

C.6 Day of the Month Calculations

This function returns the next day from the date *pDateString* parameter passed into the function.

```
Create or Replace FUNCTION getDay(pDateString in varchar2,date_period in varchar2) return
varchar2
IS
BEGIN
    Return to_char(to_date(pDateString),date_period);
End getDay;
```

The following SQL statement may be used to validate the procedure:

```
SQL> select getDay('24-FEB-2004','DAY') from dual;

GETDAY('24-FEB-2004','DAY')
-----
TUESDAY
```

C.7 Round To First Day of Month

This function rounds the date entered within *pDateString* to the first day of the month.

```
Create or Replace FUNCTION roundDate(pDateString in varchar2,date_period in varchar2)
return date
IS
BEGIN
    Return round(to_date(pDateString),date_period);
End roundDate;
```

The following SQL statements may be used to validate the procedure. The first statement shows the methods for rounding upward to the first day of the month. The second statement shows the converse operation.

```
Round Up To First Day of Month:
SQL> select select roundDate('24-FEB-2004','MONTH') from dual;

ROUNDDATE('24-FEB-2004','MONTH')
-----
01-MAR-04

Next, Round Down To First Day of Month
SQL> select roundDate('6-FEB-2004','MONTH') from dual;

ROUNDDATE('6-FEB-2004','MONTH')
-----
01-FEB-04
```

C.8 Round Day and Hours

This function returns the next day or hour from the date *pDateString* parameter passed into the function. The PM and AM suffix is applied to the final result.

```
Create or Replace
FUNCTION roundToDay(pDateString in
varchar2 default sysdate, hours in varchar default '12',
minutes in varchar2 default '00', seconds in varchar2 default '00',
amPm in varchar2 default 'PM', fcnType in varchar2 default 'DD') return varchar2
IS
BEGIN
Return to char(round(to_date(pDateString||'
'||hours||':'||minutes||':'||seconds||amPm||','
'DD-MM-YY HH:MI:SS AM'), fcnType), 'DD-MON-YY HH:MI AM');
End roundToDay;
/
```

The following SQL statements may be used to validate the procedure. This utility may be used to round day-dates to the nearest hour.

Round To Nearest Day

```
select roundToDay('24-FEB-2004','12','01','01','PM','DD') from dual;
```

```
ROUNDTODAY('24-FEB-2004','12','01','01','PM','DD')
```

```
-----
25-FEB-04 12:00 AM
```

Round To Nearest Hour

```
select roundToDay('24-FEB-2004','12','01','01','PM','HH') from dual;
```

```
ROUNDTODAY('24-FEB-2004','12','01','01','PM','HH')
```

```
-----
24-FEB-04 12:00 PM
```

Appendix D: Productivity Functions

In this section, we examine the basic productivity tools that have been written in SQL to accelerate the processing of string, character, and numeric data. The tools include basic processing algorithms for the management of geometric data, and ancillary information that contains spatial records. The functions are used by MOD within the CTD and IAS facilities for the management of geo-positioning data.

D.1 Occurrence Counting

This simple function counts the incident of the string *p_char* that occurs in the string *p_data*.

```
CREATE OR REPLACE FUNCTION count_char (
  p_data  VARCHAR2
  ,p_char  VARCHAR2 DEFAULT ' '
)
RETURN VARCHAR2 IS
  v_count  NUMBER := 0;
BEGIN
  FOR i IN 1 .. LENGTH (p_data) LOOP
    IF substr (p_data, i, length(p_char)) = p_char THEN
      v_count := v_count + 1;
    END IF;
  END LOOP;

  RETURN v_count;
END;
/
```

The following SQL statements may be used to validate the procedure. This utility is counting the characters in the demonstration string:

```
1* select count_char('Now is the time for all good men to come to the aid of their
party','the')
SQL> /

COUNT_CHAR('NOWISTHETIMEFORALLGOODMENTOCOMETOTHEAIDOFTHEIRPARTY','THE')
-----
3
SQL>
```

D.2 Word Counting

This simple function counts the words found within the input string *str*:

```
CREATE OR REPLACE FUNCTION wordcount (str IN VARCHAR2)
RETURN PLS_INTEGER
AS
  words PLS_INTEGER := 0;
  len PLS_INTEGER := NVL(LENGTH(str),0);
  inside_a_word BOOLEAN;
BEGIN
  FOR i IN 1..len + 1
  LOOP
    IF ASCII(SUBSTR(str, i, 1)) < 33 OR i > len
    THEN
      IF inside_a_word
      THEN
        words := words + 1;
        inside_a_word := FALSE;
      END IF;
    END IF;
  END LOOP;
  RETURN words;
END;
```

```

        END IF;
    ELSE
        inside_a_word := TRUE;
    END IF;
END LOOP;
RETURN words;
END;
/

```

The following SQL statements may be used to validate the procedure. This utility is counting the separate words in the demonstration string:

```

1* select wordcount('This is a sentence with 7 words') from dual;

WORDCOUNT('THISISASENTANCEWITH7WORDS')
-----
7

```

D.3 Describe Numbers

This function converts an input number *num* to a string representation of the number written as text.

```

CREATE OR REPLACE FUNCTION NUMBER_CONVERSION(NUM NUMBER) RETURN VARCHAR2
IS
    A VARCHAR2(1000);
    B VARCHAR2(20);
    X NUMBER;
    Y NUMBER := 1;
    Z NUMBER;
    LSIGN NUMBER;
    NO NUMBER;
BEGIN
    X:= INSTR(NUM, '.');
    LSIGN := SIGN(NUM);
    NO := ABS(NUM);
    IF X = 0 THEN
        SELECT          TO_CHAR(TO_DATE(NO, 'J'), 'JSP')   INTO A FROM DUAL;
    ELSE
        SELECT          to_char(to_date(SUBSTR(NO, 1,
            NVL(INSTR(NO, '.')-1, LENGTH(NO))),
            'J'), 'JSP') INTO A FROM DUAL;
        SELECT LENGTH(SUBSTR(NO, INSTR(NO, '.')+1)) INTO Z FROM DUAL;
        A := A || ' POINT ';
        WHILE Y< Z+1 LOOP
            SELECT TO_CHAR(TO_DATE(SUBSTR(NO, (INSTR(NO, '.')+Y), 1), 'J'), 'JSP')
                INTO B FROM DUAL;123
            A := A || B || ' ';
            Y :=Y+1;
        END LOOP;
    END IF;
    IF LSIGN = -1 THEN
        RETURN 'NEGATIVE ' || A;
    ELSE
        RETURN A;
    END IF;
END;
/

```

The following SQL statements may be used to validate the procedure. This utility is describing the number using alphanumeric descriptions:

```

SQL> SELECT NUMBER_CONVERSION(123) FROM DUAL;

NUMBER_CONVERSION(123)
-----
ONE HUNDRED TWENTY-THREE

```

D.4 Calculate Coordinates

This function returns the decimal equivalent of degrees, minutes and seconds entered as a parameter when the function is called.

```
Create or Replace FUNCTION CalculateCoordinates(pDeg    NUMBER, pMin NUMBER, pSec NUMBER)
RETURN FLOAT IS
    returnval    FLOAT(126);
BEGIN
    IF pDeg IS NULL OR pMin IS NULL OR pSec IS NULL Then
        returnval:=0.0;
    ELSIF pDeg < 0 THEN
        returnval:=(-1 * ((-1 * pDeg) + (pMin/60) + (pSec/3600)));
    ELSE
        returnval:=(pDeg + (pMin/60) + (pSec/3600));
    END IF;
    RETURN returnval;
END CalculateCoordinates;
```

The following SQL statements may be used to validate the procedure:

```
SQL> select calculateCoordinates(45,21,13) from dual;

CALCULATECOORDINATES(45,21,13)
-----
45.3536111
```

D.5 Calculate Degrees Minutes Seconds

This function returns the degrees, minutes and seconds of a decimal coordinate passed into the process.

```
Create or Replace FUNCTION CalculateDegMinSec(pCode CHAR DEFAULT 'D', pCoord FLOAT)
RETURN NUMBER IS
    fTemp    FLOAT(126);
    nDegree   NUMBER(38,5);
    nMinute   NUMBER(38,5);
    nSecond   NUMBER(38,5);
    returnval NUMBER(38,5);
BEGIN
    IF pCoord IS NULL THEN
        RETURN 0;
    ELSE
        fTemp:=ABS(pCoord);
        --nDegree:=TO_NUMBER(TO_CHAR(fTemp, '999'));
        nDegree:=TO_NUMBER(SUBSTR(TO_CHAR(fTemp),1,INSTR(TO_CHAR(fTemp)||'.','.')));
        fTemp:=fTemp - nDegree;
        nMinute:=fTemp * 60;

        --nSecond:=ROUND((fTemp - (nMinute/60))*3600,0);
        nSecond:=nvl(ROUND((TO_NUMBER(SUBSTR(TO_CHAR(nMinute),
        INSTR(TO_CHAR(nMinute)||'.','.'))*60),0),0);
    END IF;

    IF UPPER(pCode) = 'D' Then
        returnval:=nDegree;
        If pCoord < 0 THEN
            returnval:=returnval * -1;
        END IF;
    ELSIF UPPER(pCode) = 'M' Then
        returnval:= nvl(TO_NUMBER(SUBSTR(TO_CHAR(nMinute),
        1,INSTR(TO_CHAR(nMinute)||'.','.'))),0);
    END IF;
```

```

ELSIF UPPER(pCode) = 'S' THEN
    returnval:=nSecond;
ELSE
    returnval:=0;
END IF;
RETURN returnval;
END CalculateDegMinSec;

```

The following SQL statements may be used to validate the procedure:

```

SQL> select CalculateDegMinSec('D',45.3536111) from dual;

CALCULATEDEGMINSEC('D',45.3536111)
-----
45

SQL> select CalculateDegMinSec('M',45.3536111) from dual;

CALCULATEDEGMINSEC('M',45.3536111)
-----
21

SQL> select CalculateDegMinSec('S',45.3536111) from dual;

CALCULATEDEGMINSEC('S',45.3536111)
-----
13

```

D.6 Selecting Rows M through N of a Result

Frequently a situation arises when a user only wants to select part of a set of information from a dataset, such as the first two rows. This example demonstrates a method to return records between a minimum and maximum row number. The variable *MAX_ROW* sets the maximum row number, and *MIN_ROW* the minimum row number, that will be displayed in the final result.

```

DECLARE
MAX_ROW NUMBER:= 2;
MIN_ROW NUMBER:=1;
stmt varchar2(1000);

type mbrCursorType is ref cursor;
iCursor mbrCursorType;
machine_id number;
machine_name varchar2(100);
machine_cost number;

BEGIN
stmt:='select MACHINE_ID,MACHINE_NAME,MACHINE_COST from
(select a.*, rownum rnum from ( select machine_id,MACHINE_NAME,
MACHINE_COST from test2_tbl order by machine_id) a
where rownum <=:MAX_ROW )where rnum >= :MIN_ROW';

OPEN iCursor FOR stmt using MAX_ROW,MIN_ROW;

/*Format headings */
dbms_output.put_line('Machine Data');
dbms_output.put_line('-----');
dbms_output.put_line(rpad('Machine ID',20,' ')||' '||
rpad('Machine Name',30,' ')||' '||
rpad('Rockwell Hardness',30,' '));
dbms_output.put_line('-----');

LOOP
    FETCH iCursor into machine_id,machine_name,Rockwell_hardness;
    Exit when iCursor %NOTFOUND;

```



```

        dbms_output.put_line(rpad(machine_id,20,' ')||' '||
rpad(machine_name,30,' ')||' '||
rpad(machine_cost,30,' '));
END LOOP;
END;
/

```

The following SQL statements may be used to validate the procedure:

Machine Data		
Machine ID	Machine Name	Rockwell Hardness
1	Machine 1	5
2	Machine 2	6

D.7 Binary Objects (Blobs)

A *Blob* data type is used to store large binary objects that would be out of line for storage within the main database. This means that when a table has a *Blob* column, a row of data for that table contains a pointer or locator to the actual (within memory) position for the raw data. The *Blob* can be up to four gigabytes in size, and they participate fully in database transactions. In other words, any changes made to a *Blob* can be rolled back or committed along with other outstanding changes in the transaction. The following shows a simple example for a *Blob* application within the CTD materials database:

```

DROP TABLE T;

CREATE TABLE T
(
  NAME VARCHAR2(100),
  LOB BLOB,
  LOADTIME DATE
)

CREATE OR REPLACE PROCEDURE writeDataToLOB_proc( p_name in
varchar2,
                                                    p_buffer in RAW )
IS
  Lob_loc          BLOB;
BEGIN
  insert into t (name, lob, loadtime)
  values (p_name, empty_blob(), sysdate)
  RETURNING lob into lob_loc;

  dbms_lob.writeAppend( lob_loc, utl_raw.length(p_buffer), p_buffer );
end;
/

CREATE OR REPLACE PROCEDURE ReadDataFromLOB_proc( p_name in
varchar2 )
IS
  Lob_loc          BLOB;
BEGIN
  SELECT lob INTO Lob_loc
  FROM t
  WHERE name = p_name;

  dbms_output.put_line('THE LENGTH IS: '||dbms_lob.getlength(lob_loc));

```

```

-- just print out the first 200 bytes since put_line
-- cannot do more then 255

dbms_output.put_line('the blob is read: '||
    utl_raw.cast_to_varchar2( dbms_lob.substr( lob_loc, 200, 1 ) )
);
END;
/

```

The following SQL statements may be used to validate the procedure:

```

SQL> exec writeDataToLob_proc( '1001', utl_raw.cast_to_raw( '101023' ) );
PL/SQL procedure successfully completed.

SQL> exec readDataFromLob_proc( '1001' );
THE LENGTH IS: 6
the blob is read: 101023
PL/SQL procedure successfully completed.

SQL> exec writeDataToLob_proc( '1002', utl_raw.cast_to_raw( 'Hello World' ) );
PL/SQL procedure successfully completed.

SQL> exec readDataFromLob_proc( '1001' );
THE LENGTH IS: 6
the blob is read: 101023

```

D.8 Exponential Calculations

A simple example for using exponential arithmetic within the CTD materials database. The function is commonly used with the logarithmic functions provided in the StarLIMS shell.

```
SELECT POWER (2,150) FROM DUAL;
```

The following SQL statements may be used to validate the procedure:

```

SQL> SELECT POWER(2,150) FROM DUAL;

POWER(2,150)
-----
1.4272E+45

```

Appendix E: SQL Management Functions

In this section, we examine certain SQL functions that are used by the systems administrator to correct errors and deficiencies in the RDBMS process. This includes tools that are required to select information, create ancillary tables, manage specialized reports, and organize metadata records with the data dictionary functions. This appendix also illustrates the set-commands that are required to implement these procedures within the Oracle 8.i materials database.

E.1 Process Editing – Rollback Functions

In this example, the *Rollback* function is demonstrated. The *Rollback* is used to correct some (or all) of the changes made during a specific session. The sample use of this procedure (from within Oracle 8.i) includes the following statements:

```
SQL> SELECT * FROM TEST2_TBL;
```

MACHINE_ID	MACHINE_NAME	MACHINE_COST
1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

```
SQL> INSERT INTO TEST2_TBL VALUES(4, 'Machine4', 258999);

1 row created.

SQL> ROLLBACK;

SQL> SELECT * FROM TEST2_TBL;
```

MACHINE_ID	MACHINE_NAME	MACHINE_COST
1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

Rollback complete.

E.2 Creating a new Table from within an Existing Table

This example creates a new table (from an existing table), and then populates it with data contained in the original table. This is required for basic cut-copy-paste operations and data migration processes that occur between two or more tables. The sample use of this procedure (from within Oracle 8.i) includes the following statements:

```
SQL> SELECT * FROM TEST_NEW;
SELECT * FROM TEST_NEW
      *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> CREATE TABLE TEST_NEW AS (SELECT * FROM TEST2_TBL);
```

Table created.

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM TEST_NEW;

MACHINE_ID	MACHINE_NAME	MACHINE_COST
1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

E.3 Creating a new Table from two or more Existing Tables

This example creates a new table, from more than one existing table, using the “outer-join” operations. The sample use of this procedure (from within Oracle 8.i) includes the following statements:

SQL> SELECT * FROM TEST2_TBL;

MACHINE_ID	MACHINE_NAME	MACHINE_COST
1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

SQL> SELECT * FROM TEST4_TBL;

MACHINE_ID	MACHINE_RETIRED
1	15-JAN-04
2	28-JAN-04

```
CREATE TABLE TEST_NEW AS (SELECT tes.machine_id, tes.machine_name, tes.machine_cost,
t2.machine_retired
FROM test2_tbl tes, test4_tbl t2
WHERE ((tes.machine_id = t2.machine_id(+))));
```

The following SQL statements may be used to validate the selected operations:

SQL> SELECT * FROM TEST_NEW;

MACHINE_ID	MACHINE_NAME	MACHINE_COST	MACHINE_RETIRED
1	Machine 1	150000	15-JAN-04
2	Machine 2	160000	28-JAN-04
3	Machine 3	170000	29-JAN-04

Appendix F: SQL Plus Functions

In this section, we examine the SQL functions that are included in the SQL-Plus Oracle Package. These functions assist the MOD in their management of metadata records, including transcript procedures that are used for the chain-of-custody modeling. The examples include data dictionary terms, and set commands, that are required to hold the SQL statements for future examination at the systems administration level.

F.1 Parameters for the Data Dictionary

The following variables and parameters are used to define the data dictionary within SQL-Plus:

USER_TABLES	All tables with their name, number of columns, storage information and statistical information
USER_CATALOG	Tables, views and synonyms
USER_COL_COMMENTS	Comments on columns
USER_CONSTRAINTS	Constraint definitions for tables
USER_INDEXES	All information about indexes created for tables
USER_OBJECTS	All database objects owned by the user
USER_TAB_COLUMNS	Columns of the tables and views owned by the user
USER_TAB_COMMENTS	Comments on tables and views
USER_TRIGGERS	Triggers defined by the user
USER_USERS	Information about the current user
USER_VIEWS	Views defined by the user

F.2 Creating Log Files with Spool Functions

To send an SQL query (log) to a file use the *spool* <file> command. All information displayed on the screen is then stored in <file>, which automatically gets the first extension (as the naming convention). The command *spool off* turns all spooling off. The following SQL statements may be used to demonstrate the spooling functions:

```
SQL> spool c:\test2_tbl.dat
SQL> select machine_id as ID, machine_name as NAME,
machine_cost as COST2 from test2_tbl;
```

ID	NAME	COST
1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

```
SQL> spool off

File has been written to c:\test2_tbl.dat.
To format the file to a more easily readable format:

select machine_id as ID, machine_name as NAME,
to_char(machine_cost,'$9G999G999G999') as COST from test2_tbl;
```

```
SQL> spool c:\test2_tbl.dat
SQL> select machine_id as ID, machine_name as NAME,
```

Reference: N62558-02-C-9041

```

2  to_char(machine_cost,'$9G999G999G999')
3  as COST from test2_tbl;

```

ID	NAME	COST
1	Machine 1	\$150,000
2	Machine 2	\$160,000
3	Machine 3	\$170,000

```
SQL> spool off
```

F.3 The Set Command

With the *Set* command, the current settings of the SQL*Plus environment (for the current operating session) can be modified at the systems administration level. The following list shows the syntax and description of the most frequent settings:

COMMAND	DESCRIPTION
ECHO {OFF ON}	Determines if commands of an SQL script are displayed as they run
FEED[BACK] {6 n ON OFF}	States how many lines are returned from a query.
HEA[DING] {ON OFF}	Turns column headings on or off.
LIN[ESIZE] {80 n}	Determines the number of characters per line
NUMF[ORMAT] <format>	Sets the default format for displaying numbers
PAGES[IZE] {24 n}	Sets the number of lines per page.
PAUSE {ON OFF}	Pauses the display after each page is displayed
SPACE <number>	Set number of spaces between columns displayed
TAB {OFF ON}	Tabs are used to display the results
TERM[OUT] {ON OFF}	Controls output to the display
TRIMS[POOL] {ON OFF}	Removes trailing spaces at the end of lines in the spool file.
VER[IFY] {ON OFF}	SQL commands will not signal success to the display if off.

The following SQL statements may be used to validate the selected operations:

```
SQL> set pause on;
```

Display one screen at a time. Display will pause and wait for user to press the enter key to continue.

```
SQL> set head off;
```

```
SQL> select * from test2_tbl;
```

1	Machine 1	150000
2	Machine 2	160000
3	Machine 3	170000

Appendix G: Security Package

In this section, we provide the foundation procedures for creating user accounts and password privileges on the LIMS system, and within the CTD materials database. These procedures are accessible at the root privilege level, and cannot be adjusted by the Level I or Level II analyst. The procedures are used to maintain security throughout the CTD complex, and manage new field engineers that enter or exit the facility. The security packages use *hash* values to store all passwords -- assuring that no user can discover the phrase by accessing the database directly. Modifications can be added to this package to establish user roles, system counters, and other LIMS features.

Three tables are used with this package. The *users_tbl* stores the user information and the hash value (for the password). The privilege table *privilege lk* stores roles that can be assigned to users. The *session table session_tbl* holds active users who are currently logged into the system.

```

CREATE TABLE USERS_TBL
(
  USERNAME      VARCHAR2(50 BYTE)                NOT NULL,
  FULL_NAME     VARCHAR2(100 BYTE),
  PRIV          INTEGER                          DEFAULT 0,
  KEY_VALUE     VARCHAR2(2048 BYTE),
  EMAIL_ADDRESS VARCHAR2(50 BYTE),
  ORGANIZATION  VARCHAR2(50 BYTE),
  CREATION_DATE DATE,
  ACCESSED      NUMBER                          DEFAULT 0
)

CREATE TABLE PRIVLEDGE_LK
(
  PRIV_CODE  NUMBER                NOT NULL,
  PRIV_VALUE VARCHAR2(50 BYTE)
)

CREATE TABLE SESSION_TBL
(
  SESSION_ID NUMBER,
  TIMESTAMP  DATE,
  USERNAME   VARCHAR2(40 BYTE)
)

CREATE OR REPLACE PACKAGE security IS
--*****

CSSFile      VARCHAR2(50) := 'docman.css';
graphicspath VARCHAR2(50) := 'graphics/';
Users        VARCHAR2(20) := 'USERS_TBL';
Session_Table VARCHAR2(50) := 'SESSION_TBL';
--*****

PROCEDURE CREATE_USER(
  p_username IN VARCHAR2 default null,
  p_password IN VARCHAR2 default null,
  fullname IN VARCHAR2 default null,
  priv IN int default null,
  email in varchar2 default null,
  organ in varchar2 default null
);

```

```

PROCEDURE DELETE_USER(

updatestatus in varchar2 default 'FALSE',
p_username in varchar2 default null

);

PROCEDURE USER_MODIFY_PASSWORD(p_username in varchar2 default null,
p_password in varchar2 default null,
p_password_verify in varchar2 default null,
updatestatus in varchar2 default 'FALSE',
message in varchar2 default null);

PROCEDURE STORE_USER( p_username IN VARCHAR2 default null,
p_password in varchar2 default null,
fullname in varchar2 default null,
priv in int default 0,
email in varchar2 default null,
organ in varchar2 default null,
submitvalue in varchar2 default null);

FUNCTION DIGEST(p_username in varchar2, p_password in varchar2 )return varchar2;
PROCEDURE VALIDATE_USER( p_username in varchar2, p_password in varchar2 );
FUNCTION authenticate_username return boolean;

PROCEDURE log_cookie( p_username in varchar2, p_password in varchar2,mapid in varchar2
default null);

FUNCTION check_permission(pro_name in varchar2) return boolean;

PROCEDURE postlogin(mapid in varchar2 default null,
p_password in boolean,
password_expired in boolean,
p_username in varchar2 default null);

PROCEDURE LOGIN (initial boolean default false,
mapid in varchar2 default null);
PROCEDURE LOGOUT;

END security;
/

CREATE OR REPLACE PACKAGE BODY security AS

PROCEDURE create_user(
p_username IN VARCHAR2 default null,
p_password IN VARCHAR2 default null,
fullname IN VARCHAR2 default null,
priv IN int default null,
email in varchar2 default null,
organ in varchar2 default null
)

IS
uname_cookie owa_cookie.cookie;
priv_cookie owa_cookie.cookie;
the_uname varchar2(50);
stmt varchar2(3000);
user_role varchar2(50);
ampersand VARCHAR2(1) := CHR(38);

CURSOR cuPRIV
IS
SELECT *
FROM PRIVLEDGE_LK
ORDER BY PRIV_VALUE ASC;

BEGIN
IF SECURITY.authenticate_username THEN

```



```

uname_cookie:= owa_cookie.get('user_name');
the_uname := uname_cookie.vals(1);
priv_cookie := owa_cookie.get('user_role');
user_role   := priv_cookie.vals(1);

--http.print('<html><head>');
http.print('</head><title>Security</title>

<script language="JavaScript1.2">
    function init(){

        }

    function returnHome(){
window.open("      ", TARGET="_self" );

    }

</script>
');
http.p('
<center>');

http.p('</center>');
utility.loadcss;
menu.displaymenu;
ui_display.FormValidation;
http.p('</head>');

http.p('<body onLoad="init()" bgcolor="#ffffff" topmargin="3"
leftmargin="3" marginwidth="1" marginheight="1" text="#000066"
href="#000066" link="#000066" alink="#000066" vlink="#000000" >
');

http.print ('<br><br><br><table colspan="2"align="center"
width="30%" border="1" cellpadding="4" cellspacing="2">');
--http.p('<tr><td align="center" class="bigtitlestart"> td</tr>');

http.print ('<form name="createuser" method="post"
action="store_user" onsubmit="return validateForm(this);">');
http.print ('<table align="center" width="30%" border="1"
cellpadding="4" cellspacing="2">');
http.print ('<tr><td nowrap class=hfld> <b>Username:</b></td><td
nowrap class=hfld><input alt="blank" type=text name=p_username
size="25"></td></tr>');
http.print ('<tr><td nowrap class=hfld> <b>Password:</b></td><td
nowrap class=hfld><input type=hidden name=p_password size="25"></td></tr>');
http.print ('<tr><td nowrap class=hfld> <b>Full Name:</b>
</td><td nowrap class=hfld><input alt="blank" type=text
name=fullname value="||fullname||" size="25"></td></tr>');
--http.print ('<tr><td nowrap> <b>priv:</b></td><td><select >');

http.p('<tr><td nowrap class=hfld> <b>Privledges:</td><td
class=hfld><select size=1 name="priv" >');

    for crPRIV in cuPRIV
    LOOP
    if crPRIV.PRIV_CODE=1 then
        http.p('<option value="||crPRIV.PRIV_CODE||">||crPRIV.PRIV_value||</option>');
    end if;
    end loop;
http.p('</select></td></tr>');

--http.print ('</table>');
http.print ('<tr><td nowrap class=hfld> <b>Email
Address:</b></td><td nowrap class=hfld colspan=2>
<input alt="email" type=text name=email value="||email||" size="35"></td></tr>');

http.print ('<tr><td nowrap class=hfld> <b>
Organization:</b></td><td nowrap class=hfld
colspan=2><input alt="blank" type=text name=organ
value="||organ||" size="35"></td></tr>');

```

```

--http.p('<table align="center" width="30%" border="1"
cellpadding="4" cellspacing="2">
http.p('<tr><td nowrap class=hfld>');
http.print ('</td><td nowrap class=hfld align=center>
<input type=submit value="Create User">
<input type=button value="    Cancel    "
onClick="returnHome();"></td>
</tr>');

http.p('</form>');
http.p('</table>');
http.print ('</body></html>');

    ELSE SECURITY.LOGIN;
    END IF;
END create_user;

PROCEDURE USER_MODIFY_PASSWORD(p_username
in varchar2 default null,

p_password in varchar2 default null,
p_password_verify in varchar2 default null,
updatestatus          in varchar2 default 'FALSE',
message               in varchar2 default null)

IS
hash_dat varchar2(2048);
stmt varchar2(4000);
BEGIN
    ui_display.FormValidation;
    utility.LoadCSS;
    http.p('<html><script>
function updateStatus(){
document.usermodify.updatestatus.value='TRUE';
}

function logout(){
parent.window.location = ''          '';
}
</script>');

if updatestatus = 'TRUE' THEN

    hash_dat:=digest( p_username, p_password );

    stmt:= 'UPDATE '||package_init.userstbl||' SET  KEY_VALUE=:hash_dat WHERE
USERNAME='''||p_username|'''';
    EXECUTE IMMEDIATE stmt USING hash_dat;
    COMMIT;
http.p('<script> setTimeout("parent.window.location = "    ", 100);</script>');

END IF;

http.p('<body bgcolor="#ffffff" topmargin="3"
leftmargin="3" marginwidth="1" marginheight="1"
text="#000066" href="#000066" link="#000066"
alink="#000066" vlink="#000000" >
');

http.print ('<br><br><br><table colspan="2"align="center"
width="30%" border="1" cellpadding="4" cellspacing="2">');
if message is not null then
    http.p('<td class=val align=center>Your password
has expired...please modify</td>');
else
    http.p('<td class=val align=center>Your
initial password has expired...please modify</td>');
end if;

http.print ('<form name="usermodify" method="post"
action="security.user_modify_password" onsubmit="updateStatus();return
validateForm(this);">');
http.print ('<table align="center" width="30%"
border="1" cellpadding="4" cellspacing="2">');
http.print ('<tr><td nowrap class=hfld> <b>Username:</b></td>

```

```

<td nowrap class=hfld colspan=2>'||p_username||'<input
alt="blank" type=hidden name=p_username
value="'||p_username||'" size="15"></td></tr>');
http.print ('<tr><td nowrap class=hfld> <b>Password:</b>
</td><td nowrap class=hfld colspan=2><input alt="pwd"
type=password name=p_password size="15"></td></tr>');
http.print ('<tr><td nowrap class=hfld> <b>Verify Password:</b>
</td><td nowrap class=hfld colspan=2><input
alt="equalTo|p_password" type=password name=p_password_verify
size="15"></td></tr>');
HTP.P('<input type=hidden name=updatestatus value="FALSE">');
HTP.P('<td class=hfld ><input type=submit value="Update
Database"></td><td class=hfld ><input type=reset value=Cancel
onClick="logout();"></td></tr>');
http.p('</body></html>');
END;

```

```

PROCEDURE delete_user(
updatestatus in varchar2 default 'FALSE',
p_username varchar2 default null

```

```

)
IS
ampersand VARCHAR2(1) := CHR(38);

```

```

fullname varchar2(100);
priv int;
priv_value varchar2(50);
pwd varchar2(50);
email varchar2(50);
organization varchar2(50);

```

```

CURSOR cuUSERS
IS
SELECT *
FROM USERS_TBL
ORDER BY FULL_NAME ASC;

CURSOR cuPRIV
IS
SELECT *
FROM PRIVLEDGE_LK
ORDER BY PRIV_VALUE ASC;

```

```

BEGIN

```

```

    ui_display.FormValidation;
    utility.LoadCSS;
    http.p('
<center>');
    menu.displaymenu;

    http.p('</center>');

    http.print('<html><head>');
    http.print('</head><title>Security</title>

<script language="JavaScript1.2">
    function init(){

    }
    function validateChoice(){

var answer=confirm("This Will Permanently
Alter The User Table...Do You Wish To Continue?");

    if(answer){
        return true;
    }else{
        return false;
    }
}
}

```

```

function returnHome(){
    window.open("      ", TARGET="_self" );
}

function delete_the_user(){
window.open("security.store_USER?p_username='||
p_username||'&submitvalue=Delete%20User",TARGET="_self");
}
</script>

<link rel=stylesheet type="xxx.CSS">

<style type="text/css"><!--
.myStyle {
    position: absolute;
    visibility: hidden;
}
//--></style>
</head>');

--UI_DISPLAY.toolbar('Modify/Delete User');
if updatestatus='FALSE' then
http.p('<body onLoad="init()" bgcolor="#ffffff" topmargin="3"
leftmargin="3" marginwidth="1" marginheight="1"
text="#000066" href="#000066" link="#000066" alink="#000066"
vlink="#000000" >
');
http.print ('<br><br><br><table colspan="2"align="center"
width="30%" border="1" cellpadding="4" cellspacing="2">');
http.print('<td class=hfld colspan=2>Choose
User To Modify Or Delete:</td></tr><tr>');
for crUSERS in cuUSERS

LOOP
http.p('<td class=hfld width=40%><br></td><td class=hfld><a
href="SECURITY.DELETE_USER?UPDATESTATUS=TRUE&p_username='||
crUSERS.username||'"
onMouseOver="status='Click To Modify or Delete User:
'||crUSERS.FULL_NAME||'";return true"
onMouseOut="status=' '";return true"
onClick=""
title="Click To Modify or
Delete User: '||crUSERS.FULL_NAME||'"
>'||crUSERS.FULL_NAME||'</a></td></tr>');
end loop;
http.p('</table>');

else
select u.full_name, u.priv,u.email_address,u.organization
into fullname,pwr,email,organization
from users_tbl u
where username=p_username;

http.p('<body onLoad="init()" bgcolor="#ffffff" topmargin="3" leftmargin="3"
marginwidth="1" marginheight="1" text="#000066" href="#000066" link="#000066"
alink="#000066" vlink="#000000" >
');

http.print ('<br><br><br><table colspan="2"align="center"
width="30%" border="1" cellpadding="4" cellspacing="2">');
--http.p('<tr><td align="center" class="bigtitlestart"><font
size="3"><b> Administration Package</b></font></td></tr>');

http.print ('<form name="createuser" method="post"
action="security.store_user" onSubmit="return validateForm(this);">');
http.print ('<table align="center" width="30%" border="1"
cellpadding="4" cellspacing="2">');
http.print ('<tr><td nowrap class=hfld> <b>Username:</b>
</td><td nowrap class=hfld colspan=2>'||p_username||'<
input alt="blank" type=hidden name=p_username
value="'||p_username||'" size="15"></td></tr>');
if p_username='kurt' or p_username='joel' then
http.print ('<tr><td nowrap class=hfld> <b>Password:</b>
</td><td nowrap class=hfld colspan=2><input

```

```

alt="blank" type=password name=p_password
value="" size="15"></td></tr>');
else
http.print ('<tr><td nowrap class=hfld>
<b>Password:</b></td><td nowrap class=hfld colspan=2>
<input alt="pwd" type=password name=p_password
value="" size="15"></td></tr>');
end if;
http.print ('<tr><td nowrap class=hfld> <b>Full Name:</b>
</td><td nowrap class=hfld colspan=2><input
alt="blank" type=text name=fullname value="'||fullname||'" size="15"></td></tr>');
--http.print ('<tr><td nowrap> <b>priv:</b></td><td><select
');

http.p('<tr><td nowrap class=hfld> <b>Privledges:</td>
<td class=hfld colspan=2><select size=1 name="priv" >');

    for crPRIV in cuPRIV
    LOOP
        if crpriv.priv_value=priv_value then
            http.p('<option value="'||crPRIV.PRIV_CODE||'"
selected>'||crPRIV.PRIV_value||'</option>');
        else
            http.p('<option
value="'||crPRIV.PRIV_CODE||'">'||crPRIV.PRIV_value||'</option>');
            end if;
        end loop;
        http.p('</select></td></tr>');

http.print ('<tr><td nowrap class=hfld> <b>Email
Address:</b></td><td nowrap class=hfld colspan=2>
<input alt="email" type=text name=email value=
"'||email||'" size="35"></td></tr>');

http.print ('<tr><td nowrap class=hfld> <b>
Organization:</b></td><td nowrap class=hfld
colspan=2><input alt="blank" type=text name=organ
value="'||organization||'" size="35"></td></tr>');

--http.print ('</table>');

--http.p('<table align="center" width="30%" border="1"
cellpadding="4" cellspacing="2">

http.print ('</td><td nowrap class=hfld><input
type=button name=submitvalue value="Delete User"
onClick="delete_the_user();"></td>');
http.print ('</td><td nowrap class=hfld><input
type=submit name=submitvalue value="Modify User"></td>');
http.p('<td nowrap class=hfld><input type=button
value="Cancel" onClick="returnHome();"></td></tr>');
http.p('</form>');
http.p('</table>');

end if;
http.print ('</body></html>');

END delete_user;

PROCEDURE store_user (p_username IN VARCHAR2 default null,
p_password IN VARCHAR2 default null,
fullname IN VARCHAR2 default null,
priv IN int default 0,
email in varchar2 default null,
organ in varchar2 default null,
submitvalue in varchar2 default null)
IS
hash_dat VARCHAR2(2048);
newhash varchar2(2048);

```

```

stmt          VARCHAR2(200);
nullvalue     varchar2(10) default null;
zerovalue     number default 0;

BEGIN

    hash_dat:=digest( p_username, p_password );
    newhash:=digest(p_username,'setInitialPassword');

    if submitvalue= 'Delete User' then
        DELETE FROM USERS_TBL
        WHERE USERNAME=p_USERNAME;
        commit;

    elsif submitvalue='Modify User' then

        stmt:= 'UPDATE '||USERS||' SET USERNAME=:p_username,
        FULL_NAME=:fullname, PRIV=:priv, KEY_VALUE=:hash_dat,
        EMAIL_ADDRESS=:email, ORGANIZATION=:organ
        where USERNAME=:p_username ';

        EXECUTE IMMEDIATE stmt USING p_username,
        fullname, priv, hash_dat,email,organ,p_username;
    else
        stmt:= 'INSERT INTO '||Users||'
        VALUES (:p_username, :fullname, :priv, :hash_dat,:
        email,:organ,:created,:accessed)';

        EXECUTE IMMEDIATE stmt USING lower(p_username),
        fullname, priv, newhash,email,organ,sysdate,zerovalue;
    end if;
    COMMIT;
    -- if (sql%rowcount>0) then
        http.p('<script>
            function doRedirect()
            {
                setTimeout("parent.window.location = ", 2000);
            }
        </script>');

        http.p('<body onload="doRedirect()">
        <table border=0 width=100%>
        <tr align=center>
        <td align=center><font size="4" face="verdana,arial,times"
        color="#000066"><strong>');
        if submitvalue= 'Delete User' then
            menu.displaymenu;
            http.p('<font size="4" face="verdana,arial,times"
            color="#000066"><strong> User: '||p_username||'
            has been deleted from the database.
            </strong></font></td>');
        elsif submitvalue='Modify User' then
            menu.displaymenu;
            http.p('<font size="4" face="verdana,arial,times"
            color="#000066"><strong> User: '||p_username||'
            has been modified in the database.
            </strong></font></td>');
        else
            menu.displaymenu;
            http.p('<font size="4" face="verdana,arial,times"
            color="#000066"><strong> User: '||p_username||'
            has been stored in the database.
            </strong></font></td>');
        end if;
        http.p('</tr>

    </table>
    </body>');
    -- null;
    ---- end if;
EXCEPTION

WHEN DUP_VAL_ON_INDEX THEN

```

```

http.p('<script>alert("The user '||p_username||'
has already been used. Please enter a unique user name");
</script>');

security.create_user(p_username,p_password,fullname,priv);
END store_user;

FUNCTION digest( p_username in varchar2, p_password in varchar2 )
RETURN varchar2
IS
BEGIN
RETURN ltrim( to_char( dbms utility.get_hash_value(upper(p_username)||'/'||
upper(p_password),1000000000, power(2,30) ),rpad( 'X',29,'X')||'X' ) );
END digest;

PROCEDURE validate_user( p_username in varchar2, p_password in varchar2 )
is
    l_cnt      integer;
    handle     integer;
    stmt       varchar2(200);
    l_password  varchar2(255) default digest(p_username,p_password);
begin

    stmt:= 'select count(*) into l_cnt from '||Users||'
where username = :p_username and key_value = :l_password';

    Execute Immediate stmt using p_username, l_password;

    if (sql%rowcount=1) then
        NULL;
    end if;

    if ( sql%rowcount = 0 ) then
        raise_application_error( -20001, 'Invalid username/password' );
    end if;
end validate_user;

FUNCTION authenticate_username return boolean
IS

    l_id          number;
    l_username     varchar2(30);
    session_cookie owa_cookie.cookie;
    stmt          varchar(200);
    thesysdate     date;

BEGIN

--IF security.check_permission(owa_util.get_procedure) THEN
-- http.p('Permissions OK');

select sysdate into thesysdate from dual;
-- get the cookie value into l_id
session_cookie:= owa_cookie.get('session_id');

IF session_cookie.num_vals < 1 then
    --SECURITY.LOGIN;
    return false;

ELSE
    l_id:=session_cookie.vals(1);

    STMT:='UPDATE '||session_table||' set timestamp = :sdate
WHERE session id = :l_id AND timestamp > (:sdate - 1/24)';
    Execute Immediate STMT using thesysdate, l_id, thesysdate;
    --returning username INTO L_USERNAME; -- expire the session after
    one hour of inactivity

```

```

        IF ( sql%rowcount = 0 ) THEN

http.p('</head><table align="center"><tr><td><font
color=red><b>Your login information is either
incorrect or you session has expired</b></font>
</td></tr><br>

<tr align=center><td><font color=white><b>Please
re-enter your user information again</b></font></td></tr></html>');
http.p('<meta HTTP-EQUIV="REFRESH" CONTENT="60;
URL='||package_init.login||'>');

        return FALSE;
    ELSE
        return TRUE;
    END IF;

END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN TRUE;

END authenticate_username;

PROCEDURE log_cookie( p_username in varchar2,
p_password in varchar2,mapid in varchar2 default null)
IS
    l_cnt                number;
    l_password            varchar(255) default digest(p_username,p_password);
    l_session_id          number;
    l_user_role           number;
    stmt                 varchar2(200);
    type theRefCursorType is Ref Cursor;
    iCursor               theRefCursorType;
    theSysDate            date;
    passwordchange         boolean;
    initial_password      number;
    userdate              date;
    passwordexpired       boolean default false;

BEGIN
    -- validate the username/password is correct here...
    begin

        select creation_date into userdate from users_tbl
        where username=''||p_username||';
        exception when no_data_found then
            null;
        end;

        if sysdate>(userdate+package_init.password_expire_days) then

            passwordexpired:=true;
        end if;
        --if ( it is correct )

        if p_password=''||package_init.initialpassword||' then
            passwordchange:=true;
        end if;

        Select sysdate into theSysDate from dual;
        --http.p(mapid);
        STMT := 'select count(*) from '||users||' where
        username = :p_username and key_value = :l_password';
        open iCursor for stmt using p_username, l_password;
        Fetch iCursor into l_cnt;
        Close iCursor;

        IF (l_cnt=0) THEN

            http.print('<table width="100%" align="center"><tr>

```



```

        <td align="center"><font color="red">

        <b>Your username and password has failed to match the database –
        please try logging in again</b></font></tr></td><br>

        <tr><td align="center"><font color="white">If you
        continue to have problems logging in please contact the
        administrator</font></td></tr></table>');

    ELSE

        --Select the users privlage into a variable
        stmt:='select priv from '||users||' where username = :p_username';
        open iCursor for stmt using p_username;
        fetch iCursor into l_user_role;
        close iCursor;

        select to_char(dbms_random.value(1,35)) INTO l_session_id FROM dual;

        -- set the cookie to be l_session_id
        stmt:= 'INSERT INTO '||session_table||' VALUES
        ( :l_session_id, :thedata, :p_username)';
        execute immediate stmt using l_session_id, thesysdate, p_username;

        stmt:= 'UPDATE '||USERS||' SET ACCESSED=ACCESSED+1
        WHERE USERNAME='''||p_username||'''';
        EXECUTE IMMEDIATE STMT;
        commit;

        OWA_UTIL.mime_header('text/html', FALSE);
        OWA_COOKIE.send('session_id',l_session_id);
        OWA_COOKIE.send('user_role', l_user_role);
        OWA_COOKIE.send('user_name', p_username);
        OWA_UTIL.http_header_close;

        --Once vlidated the user is directed to display iap
        SECURITY.POSTLOGIN(mapid,passwordchange,passwordexpired,p_username);

    END IF;

END LOG_COOKIE;

function check_permission(pro_name in varchar2) return boolean
IS
    uname_cookie          owa_cookie.cookie;
    PRIV_cookie           owa_cookie.cookie;
    the_uname             varchar2(30);
    the_role              varchar2(30);
    l_id                 number;
    l_username            varchar2(30);
    session_cookie        owa_cookie.cookie;
    stmt                 varchar(200);
    thesysdate            date;
    pro_permission        integer;
    ctr                  integer default 0;

BEGIN
    --IF SECURITY.authenticate_username THEN

        uname_cookie:= owa_cookie.get('user_name');
        the_uname := uname_cookie.vals(1);
        priv_cookie := owa_cookie.get('user_role');
        the_role := priv_cookie.vals(1);

        if ctr=0 then
            return true;
        else

```

```

        return false;
    end if;

    --          ELSE SECURITY.LOGIN;
    --      END IF;

end check_permission;

PROCEDURE postlogin(mapid in varchar2 default null,
p_password in boolean,
password_expired in boolean,
p_username in varchar2 default null) is
user_priv integer default null;

BEGIN

--gather user privilage info from the cookie
http.p('<html><head>');
if p_password or password_expired then
if password_expired then
    http.p('<meta HTTP-EQUIV="REFRESH" CONTENT="0;
URL='||package_init.schema||'.security.user_modify_password?p_username='||p_username||'&me
ssage=expire">');
else
    http.p('<meta HTTP-EQUIV="REFRESH" CONTENT="0;
URL='||package_init.schema||'.security.user_modify_password?p_username='||p_username||' ">
');
end if;
else
select priv
into user_priv
from users_tbl
where username=p_username;

if user_priv =0 or user_priv=1 then
http.p('<meta HTTP-EQUIV="REFRESH" CONTENT="0;
URL='||package_init.schema||'.ui_display.display">');
elsif user_priv=2 then
http.p('<meta HTTP-EQUIV="REFRESH" CONTENT="0;
URL='||package_init.schema||'.picture_book.display">');

else
null;

end if;

end if;
http.p('</head></html>');

END postlogin;

PROCEDURE LOGIN (initial boolean default false,
                mapid in varchar2 default null)
IS
ampersand VARCHAR2(1) := CHR(38);
BEGIN
ui_display.FormValidation;
IF INITIAL THEN

http.p('<script> function setFocus(){
                                document.login.p_username.focus();
                                }
</script>');

utility.LoadCSS;
--http.p(mapid);
http.p('<body onLoad="setFocus();">');
http.print ('<table colspan="2" valign ="center"
align="center" width="200" border="0" cellpadding="4"

```

```

cellspacing="2">');
--http.print ('<tr><td colspan="2"><font
size="3"><b>||package_init.InstName||</b></font></td></tr>');
http.print ('<form name="login" method="post"
action='||package_init.Schema||'.security.log_cookie>');
http.print ('<tr><td nowrap> <b>Username:</b></td><td
align="center"><input type=text name=p_username
size="15"></td></tr>');
http.print ('<tr><td nowrap> <b>Password:</b></td><td
align="center"><input type=password name=p_password
size="15"></td></tr>');
http.p('<input type=hidden name=mapid value='||mapid||'>');
http.p('<td colspan=2 align=center>');
http.print ('</td></tr>');
http.p('<div id="f_sub" style='visibility:visible;
position:absolute;top:0;left:0;'><input type=submit></div>');

http.print ('</form>');
http.print ('</table></div>');
http.print ('</body></html>');

ELSE
--IF security.check_permission(owa_util.get_procedure) THEN
--http.print ('<html><head>');
--http.print ('<link rel=stylesheet type="text/css"
href='||package_init.servername||'/'||package_init.cssfile||'></head>');
--http.print ('</head><title></title>');
http.p('<script> function setFocus(){
document.login.p_username.focus();
}
</script>');

http.p('<body onLoad="setFocus();">');

http.print ('<table colspan="2" valign="center"
align="center" width="200" border="0" cellpadding="4" cellspacing="2">');
--http.print ('<tr><td colspan="2"><font
size="3"><b>||package_init.InstName||</b></font></td></tr>');
http.print ('<form name="login" method="post"
action='||package_init.Schema||'.security.log_cookie>');
http.print ('<tr><td nowrap> <b>Username:</b></td><td
align="center"><input type=text name=p_username size="15"></td></tr>');
http.print ('<tr><td nowrap> <b>Password:</b></td><td
align="center"><input type=password name=p_password size="15"></td></tr>');
http.p('<td colspan=2 align=center>');
http.print ('</td></tr>');
http.p('<div id="f_sub" style='visibility:visible;
position:absolute;top:0;left:0;'><input type=submit></div>');
http.print ('</form>');
http.print ('</table></div>');
http.print ('</body></html>');

END IF;
END LOGIN;

PROCEDURE LOGOUT
IS

sess_Cookie          owa_cookie.cookie;
thesession            number;
stmt                  varchar(2000);

BEGIN

    sess_cookie:= owa_cookie.get('session_id');
    thesession:= sess_cookie.vals(1);

    stmt:='delete from session_tbl where session_id = :thesession';
    execute immediate stmt using thesession;
    commit;
    ui_display.INTRO;

END LOGOUT;
-----
END security;
/

```

The following SQL statements may be used to apply the security methods. As shown, cookies are used on each workstation to facilitate data login operations (for registered users). All security tools work with Microsoft Windows and Unix operating systems. Their design allows the user to set privileges to a level that far exceeds the usual standards provided under windows for user access.

```
IS
uname_cookie          owa_cookie.cookie;
the_uname              varchar2(30);

BEGIN
  IF SECURITY.authenticate_username THEN

      uname_cookie:= owa_cookie.get('user_name');
      the_uname := uname_cookie.vals(1);

      <procedure>

  ELSE SECURITY.LOGIN;
  END IF;
```

The security procedures have been requested by MOD for their case specific operations. It is understood that MOD will use these tools to develop additional security requirements (new keys) that may be used for secure field operations. These modifications will be conducted by MOD based upon the new and emerging security requirements for database access within the 12th Main Directorate.

Appendix H: Screen Images and Laboratory Demonstrations for the CTD LIMS

In this section, we provide a series of images and screen views that demonstrate the working LIMS system. This documentation includes case examples that have been approved by the MOD 12th Main Directorate in Moscow for distribution to DTRA on 26 March 2004.

The MOD staff scientists wrote the discussion with assistance from Intek engineers in St. Petersburg. For this reason, the explanation and English usage may be difficult. In addition, certain phrases from the original text have been removed or edited based upon the specific security requirements for the scientific research facility.

In this regard, the appendix is a pre-release for the complete photographic documentation that will be presented to DTRA by CPT. Devochkin and COL. Kochin during the next delegation visit to St. Petersburg. This will also include a walk-through technical demonstration for all systems shown in this Appendix.